Generalization of Boosting Algorithms and Applications of Bayesian Inference for Massive Datasets

Gregory Kirk Ridgeway

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Washington

1999

Program Authorized to Offer Degree: Statistics

University of Washington

Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Gregory Kirk Ridgeway

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Co-Chairs of Supervisory Committee:

David Madigan

Thomas Richardson

Reading Committee:

Werner Stuetzle

Date:

In presenting this dissertation in partial fulfillment of the requirements for the Doctorial degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this thesis is allowable only for scholary purposes, consistant with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to University Microfilms, 1490 Eisenhower Place, P.O. Box 975, Ann Arbor, MI 48106, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature__________________________

Date__________________________

University of Washington

Abstract

Generalization of Boosting Algorithms and Applications of Bayesian Inference for Massive Datasets

by Gregory Kirk Ridgeway

Co-Chairs of Supervisory Committee

Professor David Madigan
Statistics

Professor Thomas Richardson
Statistics

In recent years statisticians, computational learning theorists, and engineers have developed more advance techniques to learn complex non-linear relationships from datasets. However, not only have models increased in complexity, but also datasets have outgrown many of the computational methods for fitting the models that are in standard statistical practice. The first several sections of this dissertation show how boosting, a technique originating in computational learning theory, has wide application for learning non-linear relationships even when the datasets are potentially massive. I describe particular applications of boosting for naïve Bayes classification and regression, and exponential family and proportional hazards regression models. I also show how these methods may easily incorporate many desirable properties including robust regression, variance reduction methods, and interpretability. On both real and simulated datasets and in a variety of modeling frameworks, boosting consistently outperforms standard methods in terms of error on validation datasets.

In separate but related work, the last chapter presents ideas for utilizing Bayesian methods for inference in massive datasets. Modern Bayesian analysis relies on Monte Carlo methods for sampling from complex posterior distributions, a Bayesian hierarchical model perhaps. These methods experience tremendous slowdown in computation when the posterior distribution conditions on a large dataset and the dataset cannot be summarized in terms of a small number of sufficient statistics. I develop an adaptive importance sampling algorithm that efficiently simulates draws from a posterior distribution conditioned on a massive dataset. Subsequently, I also propose a method for approximate Bayesian inference using likelihood clustering for data reduction.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I first want thank those in my academic career who have brought me to this point. My sincere appreciation goes to Prof. David Madigan who from my first day in the Department of Statistics actively mentored me, continuously proposing new directions even during my time at Microsoft and through his sabbatical year. My appreciation also goes to Prof. Thomas Richardson for his interest, involvement, and investment in my research and a very thorough read of this dissertation. Prof. Werner Stuetzle deserves my gratitude for his endless enforcement of the graduate student work ethic. Prof. John O'Kane from the Department of Orthopaedics proposed the problem that initiated my investigation of boosting and its application to automated medical diagnosis of knee injuries. My thanks also go to others in the Department of Statistics, my fellow graduate students, without whom I would not have completed my first quarter, and Kristin Sprague who always had solutions for any situation, academic or personal... as well as another dozen stories to go along.

Dr. Steven Altschuler at Microsoft Research inspired me to work on problems with practical implications, reformed my shoddy scientific programming style, funded my research for $1\frac{1}{2}$ years, and has been a good friend. I certainly never would have considered graduate school without the support of the faculty at California Polytechnic State University in San Luis Obispo including Prof. Roxy Peck, Prof. James Daly, Prof. Linda Patton, and Prof. Jay Devore. Prof. Jay Devore told me frankly, "I would be disappointed if you did anything but go to graduate school." I greatly appreciate that affirmation. I am also grateful to Dr. Colleen Love, Mel Hunter, and Kurt Haag of the Clinical Safety Project at the Atascadero State Hospital in Atascadero, California for the immense practical training. I have never sliced such fancy bologna.

Outside of my academic life, I owe much to those who prevented academics from consuming more than a third of my time. My parents, Kirk and Linda Ridgeway, and my big sister and brother-in-law, Kelly and Tim Ridgeway-Peevyhouse, have only offered encouragement as I have pursued my degree. My fellow student, officemate, housemate, partner, and friend, Daniela Golinelli has been an endless source of generosity, complementing me where I have faults. Furthermore, my close friends Steve Bement, Thomas Larry Barnhart II, Jason Mayer, and Isaac Cline have made my stay in Seattle incredibly more entertaining than I thought it would be. And finally to my redeemer who said, “For I know the plans I have for you plans to prosper you and not to harm you, plans to give you hope and a future.” (Jeremiah 29:11)
A grant from the National Science Foundation supported this work (DMS 9704573).

Chapter 1

INTRODUCTION

This dissertation investigates some of the recent trends where the fields of statistics, machine learning, and data mining meet. These trends involve the fitting of accurate, possibly non-linear, prediction models to massive and noisy datasets. This chapter serves as an overview to the thesis and gives some background details and traditions that show the need for the research that generated this dissertation.

The largest part of this dissertation discusses and develops methods involving boosting. This chapter will provide some background on boosting some of which overlaps with the research presented in chapters 2 and 3. Chapter 2 introduces the boosted naïve Bayes classifier one of the earliest boosting algorithms to admit an interpretable form. Chapter 3 presents one of the first practical algorithms for creating boosting for regression problems, the boosted naïve Bayes regression model. Chapter 4 represents the latest in boosting methodology and expands the applicability of boosting methods to the exponential family and proportional hazards regression models.

Although the problem of analyzing massive datasets is a theme through the earlier chapters, chapter 5 considers this problem directly. In it I present two methods in development that only use analysis of small manageable portions of the dataset to compute parameter estimates.

1.1 Boosting methods

In many problem domains, combining the predictions of several models often results in a model with improved predictive performance. Boosting is one such method that has shown great promise. On the applied side, empirical studies have shown that combining models using boosting methods produces more accurate classification and regression models. These methods are extendible to the exponential family as well as proportional hazards regression models. In this thesis I show that boosting, which is still new to statistics, is widely applicable. I will introduce boosting, discuss the current state of boosting, and show how these methods connect to more standard statistical practice.

1.1.1 Introduction to boosting

The trend toward model mixing had a resurgence in economics (Bates and Granger 1969), has increased in the machine learning community, and is partially accepted in the statistics community. Researchers in these fields often had different goals in mind as they developed their methodology. Breiman (1996a) specifically wanted to reduce the variance of prediction models and proposed *bagging*. Bayesian model averaging (Madigan et al. 1996, Hoeting et al. 1999) hoped to alleviate the problems associated with model selection and to account for the uncertainty involved in model search. Other methods like stacking (Wolpert 1992) and bumping (Tibshirani and Knight 1995) attempted simply to decrease prediction bias. Researchers found that in all these cases a viable solution involved fitting several models and merging the predictions that each model produced.

Boosting is one of the latest prediction methods that proponents believed, at least initially, was the latest addition to the class of model mixing procedures. Boosting, rapidly made popular in the machine learning community, is a topic slowly making its way into mainstream statistics research. Computational learning theorists desired a method for transforming a collection of weak classifiers into one strong classifier (Schapire 1990, Freund 1995). For example, given three classifiers with misclassification rates ϵ_1, ϵ_2, and ϵ_3 where $\epsilon_i < \frac{1}{2}$ they searched for ways to combine these classifiers in such a way that the combined classifier had error $\epsilon < \min\{\epsilon_1, \epsilon_2, \epsilon_3\}$.

This work culminated in Freund and Schapire (1997) that introduced the *AdaBoost* algorithm, now commonly referred to as the *Discrete AdaBoost* algorithm. They discovered an algorithm that sequentially fits "weak" classifiers to different weightings of the observations in a dataset. Those observations that the previous classifier poorly predicts receive greater weight on the next iteration. The final *AdaBoost* classifier is a weighted average of all the weak classifiers where each weak classifier receives weight dependent upon its accuracy. Many variants now exist for the *AdaBoost* algorithm, but the one with the tightest upper bound on misclassification error in Freund and Schapire's original work is as follows.

Initialize the weights of each observation to $w_i^{(1)} = \frac{1}{N}$. For t in 1 to T iterate the following steps.

1. Learn model $H_t(x_i) : X \longrightarrow [0, 1]$.

2. Compute $\epsilon_t = \sum_{i=1}^{N} w_i^{(i)} |y_i - H_t(x_i)|$ as the error for the model H_t.

3. Let $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.

4. Update the observation weights as $w_i^{(t+1)} = w_i^{(t)} \beta_t^{1-|y_i - H_t(x_i)|}$.

5. Normalize w_i^{t+1} so that they sum to one.

When the classifier performs better than random guessing on the weighted dataset $\beta_t < 1$. If $y_i = H_t(x_i)$ then the observation weight reduces by a factor of β_t. If $|y_i - H_t(x_i)| = 1$, or completely disagree, then the observation's weight does not change. For any other extent of disagreement, $0 < |y_i - H_t(x_i)| < 1$, the weight multiplier varies between β_t and 1. When $\beta_t > 1$, or the classifier performs worse than random guessing, the well-predicted observations receive more weight. After normalizing the weights this has the effect of upweighting the poorly predicted observations and downweighting the well predicted ones. The final boosted model suggested by Freund and Schapire that combines the models of each iteration is

$$H(x) = \frac{1}{1 + \prod_{t=1}^{T} \beta_t^{2r(x)-1}} \qquad \text{where} \qquad r(x) = \frac{\sum_{t=1}^{T} \left(\log \frac{1}{\beta_t}\right) H_t(x)}{\sum_{t=1}^{T} \log \frac{1}{\beta_t}}. \tag{1.1}$$

Note that $r(x)$ is a weighted average of each of the classifiers, the more accurate classifiers receiving greater weight, and it alone may be a sensible classifier. When $\beta_t > 1$, indicating that the t^{th} classifier has performed poorly, the sign on the classifier weight is negative, in effect reversing the model's predictions. Using the sigmoid transform to compute $H(x)$ results in an upper bound on misclassification that is half that of the non-transformed classifier. $H(x)$ puts out a real value that may be interpretable as a class probability estimate. A classification to 0 or 1 will depend on whether $H(x)$ exceeds a threshold, $\frac{1}{2}$ for instance.

Freund and Schapire provide little detail on the motivation behind some of the algorithmic decisions, but I will present my understanding of the origin of these steps. The first decision was to assume that one should reweight the observations in some fashion to more heavily weight poorly predicted observations. Schapire 1990 first proved that one could "boost" three weak classifiers into one strong classifier, the scenario described earlier in this section. Freund 1995 improved upon Schapire's work but both solutions involved fitting the classifiers sequentially, each time presenting reweighted observations to the learning algorithm. The second decision is to reweight in a multiplicative fashion, $w_i^{(t+1)} = w_i^{(t)} \beta_t^{\ell(y_i, H_t(x_i))}$, where $\ell(y, H)$ is a loss function. This primarily comes from the related field of on-line learning. The weighted majority algorithm (Littlestone and Warmuth 1994), a prominent work in the field of on-line learning, combines the predictions from several fixed models or experts. As data accumulate the models or experts receive more or less weight depending on how well they predicted the latest observation. The model reweighting scheme used multiplicative weights, a scheme adopted also by *AdaBoost* to apply to the observations in the dataset. Having settled on multiplicative weight updates and the loss function $\ell(y, H) = 1 - |y - H|$ Freund and Schapire sought to find an optimal value for β_t. They were able to bound the misclassification rate on the *training* dataset of the final combined classifier to be

$$\epsilon \leq \frac{1}{2} \prod_{t=1}^{T} \frac{1 - (1 - \epsilon_t)(1 - \beta_t)}{\sqrt{\beta_t}} \tag{1.2}$$

Minimizing the upper bound 1.2 with respect to β_t results in the choice $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$. With this choice for β_t the bound becomes

$$\epsilon \leq 2^{T-1} \prod_{t=1}^{T} \sqrt{\epsilon_t(1 - \epsilon_t)} \tag{1.3}$$

Note that in 1.3 as long as the weighted misclassification rate does slightly better or slightly worse than random guessing the bound decreases. If the weighted error is always $\epsilon_t = \frac{1}{2} - \gamma$ then Freund and Schapire deduce the bound $\epsilon \leq e^{-2T\gamma^2}$ so that the misclassification rate on the training set goes exponentially fast to zero. Although in practice ϵ_t does not stay constant, this does indicate that the algorithm may fit the training dataset quickly.

Although this training speed may be disconcerting for fears of rapid overfitting, in a wide variety of classification problems, their weighting scheme and final classifier merge have proven to be an effective method for reducing bias and variance, and improving misclassification rates (Bauer and Kohavi 1999, Dietterich 1998). Empirical evidence has shown that the base classifier can be fairly simplistic (shallow classification trees) and yet, when boosted, can capture complex decision boundaries (Breiman 1998).

I began work in the area of boosting soon after Freund and Schapire (1997) appeared in print. Chapter 2 reports the findings of my earliest work on the topic. The model merging scheme in 1.1 appears complex. With the goal to combine *AdaBoost*'s performance gains while maintaining interpretability, I showed that the boosted naïve Bayes classifier can achieve both. The computational learning theorists were primarily concerned with boosting classification problems, but it is natural to consider their applicability to regression problems. Chapter 3 proposes a method of directly transferring boosting from classification to regression via a transformation of the dataset. I am able to use a variant of the interpretable boosted naïve Bayes classifier derived in chapter 2 to fit boosted regression models. The boosted naïve Bayes classification and regression work, following Charles Elkan's lead (Elkan 1997), continued to make connections between boosted models and non-linear additive models, a connection made explicit in later work. Although these methods helped extend the practicality of boosting, they did little to explain why boosting seemed to work.

From the statistician's point of view, the breakthrough in boosting occured in Friedman, Hastie, and Tibshirani (1998). Although still focused on classification problems, they showed that Freund and Schapire's *AdaBoost* algorithm is an optimization method for finding a classifier that minimizes a particular exponential loss function. Other authors have also made the connection between boosting and optimization (Breiman 1997, Mason et al. 1999). Friedman et al. proceeded to show that the exponential loss function used in *AdaBoost* is closely related to the Bernoulli likelihood and developed a new boosting algorithm that finds a classifier to directly maximize a Bernoulli likelihood. The important finding of this work is that it links the work of the computational learning theorist to a likelihood method of standard statistical practice.

Although boosting algorithms on the surface appear complex, closer inspection will show that they have much in common with how a statistician might fit familiar linear models. This similarity will lead directly into their application to exponential family and proportional hazards regression models, all of which can admit a boosted form. Also we can exploit the modular structure of boosting algorithms to ensure that our model has the most desirable statistical properties, robustness, variance reduction, interpretability, and scalability to massive datasets. Lastly, I will address the question as to whether boosting algorithms belong in the class of model mixing methods and where I believe the future of prediction models lies.

1.1.2 Boosting for classification

Even before it was published, Freund and Schapire's *Discrete AdaBoost* was generating great interest. It seemed that the performance of just about any classifier could be improved both in terms of bias and variance. Furthermore, the procedure seemed immune to overfitting. One would simply fit the classifier, $h_1(\mathbf{x})$, to the dataset, upweight the observations it misclassified, and fit another classifier, $h_2(\mathbf{x})$, using the new observation weights. The algorithm iterates several times piling more weight on the difficult to classify observations. In the end all the classifiers participate in a weighted vote for the final classification rule. Freund and Schapire developed their particular weight and merge scheme to iteratively reduce an upper bound on the training error.

Some early research investigated boosting specific types of classifiers using the *Discrete AdaBoost* algorithm. Classification trees seem to dominate most boosting research because of their flexibility and performance in high-dimensions. Furthermore, boosting tends to improve classification trees greatly since they are often unstable, slight variations in the dataset drastically changing their predictions. Leo Breiman said that AdaBoost with trees is the best "off-the-shelf" classifier (Breiman 1998).

The Knowledge Discovery and Data mining conference in 1997 (KDD'97) helped bring the topic of boosting to the attention of practical data analysis. Charles Elkan's application of boosted naïve Bayes won first place out of 45 entries in the KDD'97 data mining competition (Elkan 1997). Many of the competitors used very complex algorithms. However, the boosted naïve Bayes classifier, though comparatively simple, still outperformed them on the posed problem. My own interest in boosting began at this point and lead to my first contribution to the topic. While the usual non-boosted naïve Bayes approach leads to elegant and effective explanations (see, for example, Madigan, Mosurski, and Almond 1996 and Becker, Kohavi, and Sommerfield 1997), the *AdaBoost*-ed version destroys this feature. The research presented in chapter 2 as well as Ridgeway, Madigan, Richardson, and O'Kane (1998) showed that the boosted naïve Bayes classifier forms decision boundaries that are approximately linear in the predictors. This allows easy interpretation of the model's reasoning process.

Besides empirical exploration into the performance of these classifiers, statisticians and computational learning theorists began the search for the explanation. Freund and Schapire (1997) showed that in each iteration *AdaBoost* reduces the training error and produced some generalization error bounds based on VC dimension. However, the performance in practice outpaced this bound.

More recently attention has shifted to a refinement of the original *AdaBoost* algorithm. The *Real AdaBoost* algorithm (Schapire and Singer 1998) produces "confidence rated predictions". Suppose we wish to predict a binary outcome $Y \in \{-1, 1\}$ with a classifier $F(\mathbf{x}) \in \Re$ that puts out a positive number if it favors $Y = +1$ and a negative number if it favors $Y = -1$. The magnitude of $F(\mathbf{x})$ determines the confidence in the prediction. In detail the *Real AdaBoost* algorithm proceeds as shown in figure 1.1. It differs from the *Discrete AdaBoost* algorithm in that it performs the boosting on a log-odds scale rather than a probability scale, has a slightly different reweighting scheme, and combines the classifiers more simply.

In a rather obscure technical report, Breiman (1997) first noticed that *AdaBoost* was an optimization algorithm. This connection with optimization was clarified and expanded upon in Friedman, Hastie, and Tibshirani (1998). Their argument is as follows. As before, let $Y \in \{-1, 1\}$ and our classifier, $F(\mathbf{x})$, will produce a real-valued output. Let the loss of a particular classifier be

$$J(F) = \mathrm{E}\left[e^{-yF(\mathbf{x})}|\mathbf{x}\right]. \tag{1.4}$$

The loss function in (1.4) is small when the signs of y and $F(\mathbf{x})$ agree, or in other

Initialize the weights of each of the N observations to $w_i^{(1)} = \frac{1}{N}$. For t in $1, \ldots, T$ iterate the following steps.

1. Using the weights, estimate $P(Y = 1|\mathbf{x})$.

2. Set $h_t(\mathbf{x}_i) = \log \frac{\hat{P}(Y=1|\mathbf{x})}{\hat{P}(Y=-1|\mathbf{x})}$.

3. Update the observation weights as
   $w_i^{(t+1)} = w_i^{(t)} \exp(-\alpha_t y_i h_t(\mathbf{x_i}))$.

4. Normalize $w^{(t+1)}$ to sum to 1.

The final boosted model suggested by Freund and Schapire that combines the models of each iteration is

$$\mathrm{H}(\mathbf{x}_i) = \mathrm{sign}\left(\sum_{t=1}^{T} h_t(\mathbf{x_i})\right).$$

Here α_t is a tuning parameter that we will set to 1.

Figure 1.1: The Real AdaBoost algorithm

words, when the classification is correct. One can also show that (1.4) is an upper bound, though not a very tight one, on the misclassification rate (Schapire and Singer 1998). Given a current classifier one might wish to improve upon it by adding a new term, $f(\mathbf{x})$, so that $J(F+f) < J(F)$. Minimizing $J(F+f)$ with respect to f yields that

$$f(\mathbf{x}) = \frac{1}{2}\log\frac{E_w(Y|\mathbf{x})}{1-E_w(Y|\mathbf{x})} = \frac{1}{2}\log\frac{P_w(Y=+1|\mathbf{x})}{P_w(Y=-1|\mathbf{x})}$$

where $E_w(Y|\mathbf{x}) = \frac{E(w(Y,\mathbf{x})Y|\mathbf{x})}{E(w(Y,\mathbf{x})|\mathbf{x})}$ and $P_w(\cdot)$ is a weighted conditional probability with weights $w = e^{-yF(\mathbf{x})}$, usually estimated in practice using trees, neural networks, or naïve Bayes classifiers. Friedman et al. show that repeatedly updating using $F(\mathbf{x}) \leftarrow F(\mathbf{x}) + f(\mathbf{x})$ is equivalent to the *Real AdaBoost* algorithm.

To the statistician, minimizing the exponential bound (1.4) seems unnatural since we more often work with maximizing likelihoods. Friedman et al. (1998) noted that the minimizer of (1.4) is the same as the maximizer of the expected Bernoulli log-likelihood

$$J(F) = \mathrm{E}\ell(F) = \mathrm{E}\left[y^*F(\mathbf{x}) - \log\left(1+e^{F(\mathbf{x})}\right)|\mathbf{x}\right]. \tag{1.5}$$

where $y^* = \frac{1}{2}(1+y) \in \{0,1\}$. The expression in 1.5 assumes that $y^* \sim \mathrm{Bern}(p(\mathbf{x}))$ where $p(\mathbf{x}) = \frac{1}{1+e^{-F(\mathbf{x})}}$. Furthermore, the exponential criterion and the Bernoulli log-likelihood are equivalent to a second order Taylor expansion around $F = 0$. At this point Friedman et al. take the step to produce a boosting algorithm that directly maximizes the Bernoulli log-likelihood (1.5) rather than the exponential misclassification bound (1.4). Similar to their *Real AdaBoost* derivation, given a current classifier their *LogitBoost* algorithm adds new components to further increase the Bernoulli log-likelihood using Newton steps.

We can select one of many optimization methods to find an f that increases $J(F+f)$, the Bernoulli log-likelihood shown in (1.5). Friedman et al. chose to derive a Newton algorithm. Given a current estimate for $F(\mathbf{x})$ the Newton update for $\hat{F}(\mathbf{x})$

Table 1.1: Misclassification rates of boosted trees - from Friedman, et al (1998)

	CART	AdaBoost	LogitBoost
Breast	4.5%	4.0%	2.9%
Ion	7.6%	6.8%	7.1%
Glass	40.0%	25.7%	26.6%
Sonar	59.6%	20.2%	20.2%
Waveform	36.4%	19.5%	20.6%

is

$$\hat{F}(\mathbf{x}) \leftarrow \hat{F}(\mathbf{x}) - \frac{\frac{\partial}{\partial f} J(\hat{F}+f)|_{f=0}}{\frac{\partial^2}{\partial f^2} J(\hat{F}+f)|_{f=0}} \tag{1.6}$$

$$= \hat{F}(\mathbf{x}) + \mathrm{E}_w\left[\frac{y^* - p(\mathbf{x})}{p(\mathbf{x})(1-p(\mathbf{x}))}|\mathbf{x}\right] \tag{1.7}$$

$$\text{where } w = p(\mathbf{x})(1-p(\mathbf{x})) \text{ and } p(\mathbf{x}) = \frac{1}{1+e^{-\hat{F}(\mathbf{x})}} \tag{1.8}$$

Newton optimization, therefore, tells us to repeatedly substitute weighted expectations to improve $\hat{F}(\mathbf{x})$. Note that the weighting in (1.8) implies that the largest weights pile onto the difficult to classify observations, those near $p(\mathbf{x}) = \frac{1}{2}$. This is at least similar in spirit to the *AdaBoost* algorithm. Since we do not know the value of the expectations in (1.7) we can approximate them by choosing one of our favorite regression methods. Substituting a linear model in (1.7) for the weighted expectation reduces to the iteratively reweighted least squares algorithm for fitting linear logistic regression models. Friedman et al. use CART (Breiman et al. 1984) regression in their experimental results and show substantial improvement in accuracy over a non-boosted CART classifier. Table 1.1 shows the effect of boosting on the misclassification rate using five UCI classification datasets (Merz and Murphy 1998).

Computational learning theorists are more eager to claim boosting's association with the theory of margins and support vector machines than with optimization and likelihoods (Schapire 1999a, Schapire et al. 1998). However, these authors are also beginning to discuss the relationship to statistics (Schapire 1999b, Mason et al. 1999). But once boosting and likelihood methods become related, the door opens to many other statistical models. And so the findings of Friedman, Hastie, and Tibshirani (1998) lead us directly into methodology for generating boosting algorithms for a large class of other likelihood based models.

1.1.3 Boosting for regression

At the conclusion of their paper, Freund and Schapire (1997) outline their ideas for applying the *AdaBoost* algorithm to regression problems. However, Drucker (1997) first proposed and tested practical methods for boosting regression. His *ad hoc* method followed the spirit of the *AdaBoost* algorithm by repeatedly performing weighted tree regression followed by upweighting the poorly predicted observations and downweighting the well predicted ones. Compared with fitting just one tree, his method seems to be competitive. Breiman (1997) suggests how one might deal with boosting regression problems with his *arc-gv* methodology. These methods have not yet been investigated. His more recent work on adaptive bagging (Breiman 1999), which is quite similar to boosting ideas, appears to showing promising improvements in terms of bias and variance.

Prior to Friedman, Hastie, and Tibshirani (1998) no clear way for importing boosting to regression problems existed. Adapting some of the ideas on boosting regression from Freund and Schapire (1997), I was able to create a boosted regression algorithm. Ridgeway, Madigan, and Richardson (1999) proposed mapping the regression problem into a classification problem, applying the boosted naïve Bayes classifier proposed in chapter 2, and transforming the resulting classifier back as a regressor. They showed that this method fits an additive model but also estimates a transformation of the response variable. Chapter 3 shows the derivation of this algorithm and tests its performance with other interpretable multivariate regression procedures. The empirical results show that it performs similarly to generalized additive models (Hastie and Tibshirani 1990) and outperforms CART in four simulated examples but performs slightly worse on one of the two real datasets investigated.

Of these mentioned works related to boosting regression problems only Breiman (1997) alludes to involving optimization of a regression loss function as part of the boosting algorithm. Friedman (1999a) and the companion paper Friedman (1999b) extended the work of Friedman, Hastie, and Tibshirani (1998) and created the ground work for a new generation of boosting algorithms. Using the connection between boosting and optimization made explicit in Friedman et al. (1998), this new work proposes the Gradient Boosting Machine.

In any function estimation problem we wish to find a regression function, $\hat{F}(\mathbf{x})$, that minimizes the expectation of some loss function, $\Psi(y, F)$, as shown in (1.11).

$$\begin{aligned}\hat{F}(\mathbf{x}) &= \arg\min_{F(\mathbf{x})} \mathrm{E}_{y,\mathbf{x}} \Psi(y, F(\mathbf{x})) \\ &= \arg\min_{F(\mathbf{x})} \mathrm{E}_{\mathbf{x}} \left[\mathrm{E}_{y|\mathbf{x}} \Psi(y, F(\mathbf{x})) \middle| \mathbf{x} \right] \end{aligned} \tag{1.11}$$

We will focus on finding estimates of $F(\mathbf{x})$ such that

$$\hat{F}(\mathbf{x}) = \arg\min_{F(\mathbf{x})} \mathrm{E}_{y|\mathbf{x}} \left[\Psi(y, F(\mathbf{x})) | \mathbf{x} \right]. \tag{1.12}$$

Parametric regression models assume that $F(\mathbf{x})$ consists of a finite number of parameters, $\boldsymbol{\beta}$, and estimates them by selecting those values that minimize a loss function (i.e. squared error loss) over a training sample of N observations on $(y, \mathbf{x})$ pairs as in (1.13).

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{N} \Psi(y_i, F(\mathbf{x}_i; \boldsymbol{\beta})) \tag{1.13}$$

When we wish to estimate $F(\mathbf{x})$ non-parametrically the task becomes more difficult. Again we can proceed similarly to (1.6) and modify our current estimate of $F(\mathbf{x})$ by adding a new function $f(\mathbf{x})$ in a greedy fashion. Letting $F_i = F(\mathbf{x}_i)$, we see that we

Initialize $\hat{F}(\mathbf{x}) = \arg\min_\rho \sum_{i=1}^N \Psi(y_i, \rho)$ where ρ is a constant.

For t in $1, \cdots, T$ do

1. Compute the negative gradient as the working response

$$z_i = -\frac{\partial}{\partial F(\mathbf{x}_i)} \Psi(y_i, F(\mathbf{x}_i)) \Big|_{F(\mathbf{x}_i)=\hat{F}(\mathbf{x}_i)} \tag{1.9}$$

2. Fit a regression model, $f(\mathbf{x})$, predicting z_i from the covariates $\mathbf{x}_i$.

3. Choose a gradient descent step size as

$$\rho = \arg\min_\rho \sum_{i=1}^N \Psi(y_i, \hat{F}(\mathbf{x}_i) + \rho f(\mathbf{x}_i)) \tag{1.10}$$

4. Update the estimate of $F(\mathbf{x})$ as

$$\hat{F}(\mathbf{x}) \leftarrow \hat{F}(\mathbf{x}) + \rho f(\mathbf{x})$$

Figure 1.2: Friedman's Gradient Boost algorithm

want to decrease the N dimensional function

$$\begin{aligned} J(\mathbf{F}) &= \sum_{i=1}^{N} \Psi(y_i, F(\mathbf{x}_i)) \\ &= \sum_{i=1}^{N} \Psi(y_i, F_i). \end{aligned} \tag{1.14}$$

The negative gradient of $J(\mathbf{F})$ indicates the direction of the locally greatest decrease in $J(\mathbf{F})$. Gradient descent would then have us modify $\mathbf{F}$ as

$$\hat{\mathbf{F}} \leftarrow \hat{\mathbf{F}} - \rho \nabla J(\mathbf{F}) \tag{1.15}$$

where ρ is the size of the step along the direction of greatest descent. Clearly, this step alone is far from our desired goal. First, it only fits F at values of $\mathbf{x}$ for which we have observations. Second, it does not take into account that observations with similar $\mathbf{x}$ are likely to have similar values of $F(\mathbf{x})$ (i.e. $F(\mathbf{x})$ is likely continuous and smooth). Both these problems would have disastrous effects on generalization error. However, Friedman suggests selecting a class of functions that use the covariate information to approximate the gradient. This line of reasoning produces his Gradient Boosting algorithm shown in figure 1.2. At each iteration the algorithm determines the direction, the gradient, in which it needs to improve the fit to the data and selects a particular model from the allowable class of functions that is in most agreement with the direction.

Figure 1.3 demonstrates the geometry of the gradient boosting machine. In this case the true regression function, $F(\mathbf{x})$, is the sinusoidal curve and our current estimate is the constant line at zero, $\hat{F}(\mathbf{x}) = 0$. Assuming that we had 100 observations from some likelihood based model, we can compute the gradient of the log-likelihood at each of those 100 observations. The vertical lines indicate the gradient direction that each of these points would like to move in order to further increase the associated log-likelihood. Clearly there is randomness, but on average the gradient indicates moves closer to the true $F(\mathbf{x})$. Incorporating covariate information we can approximate the gradient by a one split CART model shown in figure 1.3 as the step

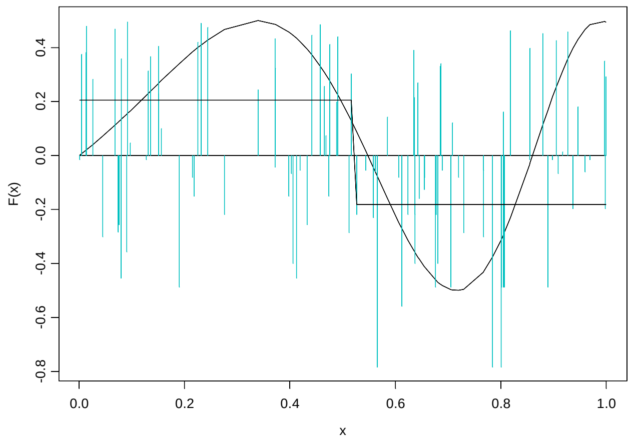

Figure 1.3: Geometry of the Gradient Boosting Machine

function. The CART model seems to capture the general trend of the gradient and makes a move from $\hat{F}(\mathbf{x}) = 0$ closer to the true $F(\mathbf{x})$. This step function would be the gradient step direction in the function space.

In the case of squared-error loss, $\Psi(y_i, F(\mathbf{x}_i)) = \sum_{i=1}^{N}(y_i - F(\mathbf{x}_i))^2$, this algorithm corresponds exactly to residual fitting. Friedman also uses this algorithm to develop new boosting algorithms for robust regression with least absolute deviation and Huber loss functions (Huber 1964).

Between the work of Friedman, Hastie, and Tibshirani (1998) and Friedman (1999a) we have two routes toward boosting algorithms for fitting models that are in common statistical practice. We can mimic the derivation of the *LogitBoost* algorithm for other likelihoods or insert likelihood based loss functions for $\Psi(y, F)$ in the gradient boosting framework. Chapter 4 explores both of these routes for fitting exponential family and proportional hazards regression models. In particular I derive a general boosting algorithm for exponential family models based on Fisher-scoring optimization. The *LogitBoost* algorithm is a special case of this class. I also derive gradient boosting algorithms for both exponential family and proportional hazards models. I show on simulated and real datasets that these methods seem to outperform more standard regression techniques.

1.1.4 Improvements on boosting

Boosting algorithms have a convenient modular structure that expose themselves to further improvements. All of the algorithms consist of an update stage of the form

$$\hat{F}(\mathbf{x}) \leftarrow \hat{F}(\mathbf{x}) + \mathrm{E}_w \left[z(y, \hat{F}(\mathbf{x}))|\mathbf{x} \right]. \tag{1.16}$$

Different likelihoods lead to different functional forms for $z(y, \hat{F}(\mathbf{x}))$ within this framework. We can also estimate the conditional expectation by the regression method of our choice, planes, smoothers, and trees for example. There are many other possible variations.

At the end of chapter 4 I discuss several ways in which the structure of the boosting algorithm may be exploited to achieve several ideal statistical properties. In particular, we can create boosting predictors that are

- more **accurate** on test data by controlling the algorithms optimization or learning rate,
- more **stable** by introducing a bagging or subsampling stage,
- **scalable** by randomly subsampling, deterministically sampling, or weight trimming,
- **robust** by replacing the conditional expectation by a robust estimator of central tendency, and
- **interpretable** by forcing low order function approximation.

Research, including this work, is now showing that boosting represents a new class of learning algorithms that Friedman correctly named "gradient machines". As all the extensions described here show, boosting iteratively fits some form of the residual, regions of the sample space where the model's predictions are missing the target data. This being the case, the original class of model mixing procedures are not in competition with boosting but rather can coexist inside or outside a boosting algorithm. That is, one could average across boosted estimates or perform bagging and bumping within a boosting iteration. The future of regression may hold new hybrid methods that are robust, have low bias and variance, and adequately account for model uncertainty.

1.2 Bayesian inference in massive datasets

The advent of the massive dataset and the expansion of the field of data mining has created the need to produce statistically sound methods that scale to these large problems. Although statistics in the past has dealt with extracting maximum information from a small dataset, these new problems require methods that can compress the massive dataset to something humanly understandable.

In chapter 5 I introduce an algorithm based on importance sampling to facilitate Monte Carlo simulations when the posterior distribution conditions on a massive dataset. The method uses a posterior distribution conditioned on a smaller, more manageable partition of the dataset as a sampling distribution. With this choice the importance weight function is simple for any readily computable likelihood function. Incorporating the entire dataset requires only one scan of the remaining observations. Derived from an adaptive importance sampling algorithm, I propose predictive weight trimming to gain further sampling efficiency.

In chapter 5 I present some preliminary ideas on a new method for performing inference in a massive dataset. When faced with a massive dataset the likely response from a statistician is to select a random sample and fit a model to the sample. The possibility exists, however, that with a little additional effort one could compose a more principled, stratified subsample that outperforms simple random sampling but avoids iterating many times over the full dataset to fit the desired model.

The principle that drives this work is that many observations in a massive dataset might be redundant, either copies or near copies of other observations. DuMouchel et al. (1999) proposed a method that builds a pseudo-dataset that matches the moments of the original dataset. They suggest that if the likelihood is smooth then "data squashing" will reduce the redundancy while preserving much of the information contained in the full dataset. Their method is independent of the model under consideration so long as the model relies only on those moments being preserved, but shows only a moderate improvement over simple random sampling.

With a likelihood-based model in mind we may be able to stratify the observations into clusters with similar likelihood, improving upon the data squashing approach as well as the simple random sample. More precisely, if many observations have the same value of the likelihood function for all values of the parameters, we would like to cluster them and carry out a likelihood based inference using a weighted likelihood of the clustered data. In all steps I try to limit the number of expensive scans of the dataset. The process of "likelihood clustering" reduces redundancy in the massive dataset producing a smaller weighted dataset for which standard statistical tools are tractable. Using a few heuristics to achieve the likelihood clustering principles I show some empirical results on a simple linear regression model. The evidence indicates that the additional effort to cluster by likelihood results in substantial improvement over a simple random sample.

1.3 Contributions of this research

1. I showed that the boosted naïve Bayes classifier forms approximately linear decision boundaries and therefore admits an interpretable form. I also show that the decision boundary is exactly linear when boosting the naïve Bayes

classifier using *Real AdaBoost*. (Chapter 2)

2. I applied the interpretable boosted naïve Bayes classifier to an application in knee injury diagnosis. The work shows that the classifier is not only accurate (88% accuracy) but also easily interpretable for the practicing sports medicine physician even in the case of missing predictors. (Chapter 2)

3. I developed one of the first practical boosted regression procedures. I show its relationship to additive models and demonstrate that it is competitive against other interpretable multivariate regression procedures. (Chapter 3)

4. I generalized the boosting methodology to fit exponential family regression models. For this purpose I derived a general boosting algorithm based on Fisher scoring of which the *LogitBoost* algorithm is a special case. I also developed a gradient ascent algorithm for the same purpose. For the gradient algorithms I show that one can use existing software for fitting generalized linear models. (Chapter 4)

5. I also extended the boosting methodology to fit proportional hazards regression models. I derived boosting algorithms for parametric survival models as well as models fit using Cox's partial likelihood. Boosting can also estimate non-parametrically both the regression function and the baseline hazard function. (Chapter 4)

6. I derived an importance sampling algorithm for sampling from a posterior distribution that conditions on a massive dataset. It has an efficient sampling distribution and features an importance weight computation that is linear in the number of observations and predictors. (Chapter 5)

7. I developed an adaptive importance sampling algorithm for sampling from a

posterior distribution that conditions on a massive dataset. It increases the efficiency of the importance sampler while maintaining an easy sampling distribution and fast weight computation. (Chapter 5)

8. I proposed the "likelihood clustering" method for compressing a massive dataset into a smaller pseudo-dataset. On a simulated simple linear regression example I show that with a little additional computation likelihood clustering greatly outperforms simple random sampling. (Chapter 5)

Chapter 2

BOOSTED NAÏVE BAYES CLASSIFICATION

Voting methods such as boosting and bagging provide substantial improvements in classification performance in many problem domains. However, the resulting predictions can prove inscrutable to end-users. This is especially problematic in domains such as medicine, where for reasons of safety and liability end-user acceptance often depends on the ability of a classifier to explain its reasoning. Here I propose a variant of the boosted naïve Bayes classifier that facilitates explanations while retaining predictive performance. These results may also be found in Ridgeway, Madigan, Richardson, and O'Kane (1998).

2.1 Introduction

In classification problems we have a set of predictors or features with which we wish to accurately predict a response or target that takes on a discrete, finite value. In this chapter we will focus on binary classification problems, situations in which our goal is to construct a function of our predictors, $h(x)$, that accurately predicts our response $Y \in \{0, 1\}$. Traditional statistical practice often appeals to methods such as linear discriminant analysis or linear logistic regression although methods such as additive logistic regression (Hastie and Tibshirani 1990), neural networks (Bishop 1995), and tree-based classifiers such as CART (Breiman, Friedman, Olshen, and Stone 1984) are seeing more use.

Efforts to develop classifiers with strong discrimination power using voting methods do not stress the importance of comprehensibility. Bauer and Kohavi (1999) state that "for learning tasks where comprehensibility is not crucial, voting methods are extremely useful." However, as many authors have pointed out, problem domains, such as credit approval and medical diagnosis, do require interpretable as well as accurate classification methods. For instance, Swartout (1983) commented that "trust in a system is developed not only by the quality of the results but also by clear description of how they were derived. ... In addition to providing diagnoses or prescriptions, a consultant program must be able to explain what it is doing and why it is doing it." In this chapter I present a boosted naïve Bayes classifier with both competitive discrimination ability and transparent reasoning. I describe the proposed boosted, interpretable naïve Bayes classifier, show how it relates to the variants of *AdaBoost*, and examine its performance empirically.

2.2 The naïve Bayes classifier

Probabilistic classifiers for two-class problems take the form $P(Y = y|\mathbf{X})$. Using Bayes' theorem we can express this as

$$P(Y = y|\mathbf{X}) = \frac{P(Y = y)P(\mathbf{X}|Y = y)}{P(\mathbf{X})} \tag{2.1}$$

The naïve, simple, or idiot's Bayes assumption is that the components of $\mathbf{X}$ are conditionally independent given the class label. This assumption permits the factorization of the second term in the numerator of 2.1.

$$P(Y = y|\mathbf{X}) = \frac{P(Y = y)\prod_{j=1}^{d} P(X_j|Y = y)}{P(\mathbf{X})} \tag{2.2}$$

Under the naïve Bayes assumption 2.2, writing the log-odds in favor of $Y = 1$ we obtain

$$\begin{aligned}\log\frac{P(Y = 1|X)}{P(Y = 0|X)} &= \log\frac{P(Y = 1)}{P(Y = 0)} + \sum_{j=1}^{d}\log\frac{P(X_j|Y = 1)}{P(X_j|Y = 0)} \\ &= w_0 + \sum_{j=1}^{d} w_j(X_j) \end{aligned} \tag{2.3}$$

The w_j are the weights of evidence described by Good (1965). A positive $w_j(X_j)$ indicates that the state of X_j is evidence in favor of the hypothesis that $Y = 1$. A negative weight is evidence for $Y = 0$. Spiegelhalter and Knill-Jones (Spiegelhalter and Knill-Jones 1984) advocate the use of weights of evidence extensively in medical diagnosis and propose evidence balance sheets as a means of viewing the reasoning process of the naïve Bayes classifier. Madigan, Mosurski, and Almond (1996) and Becker, Kohavi, and Sommerfield (1997) develop this idea further.

The expression in 2.3 has a striking similarity to the usual logistic regression model. These two models, however, are based on different assumptions, the naïve Bayes assumption being more restrictive. Logistic regression assumes that $P(Y = 1|x) = P(Y = 0|x)\exp(\sum h_j(x_j))$ where $P(Y = 0|x)$ is an arbitrary baseline function. The log-odds then have the form of 2.3. The naïve Bayes assumption, on the other hand, assumes that $P(Y = 0|x) = \exp(\sum g_j^{(0)}(x_j))$ and $P(Y = 1|x) = \exp(\sum g_j^{(1)}(x_j))$ so that *both* conditional class probabilities are exponentials of an additive function of the predictors. In either case the log-odds have the form

$$\log \frac{P(Y = 1|x)}{P(Y = 0|x)} = \sum_{j=1}^{d} w_j(x_j). \tag{2.4}$$

Even though in the end the models have the same form, these assumptions lead to different estimation procedures. Linear logistic regression estimation is based on direct maximum likelihood estimates of the w_j's and involves iteratively weighted least squares. Each iteration of the least squares algorithm requires a scan of the dataset and a $d \times d$ matrix inversion.

Estimation of the naïve Bayes model first estimates the conditional probabilities $(P(Y = 1), P(Y = 0), P(X_j|Y = 0), P(X_j|Y = 1)$, etc.) via maximum likelihood and then computes the w_j's as the respective log-odds. Although the assumptions are more restrictive they lead to two convenient properties. The first attractive feature of this process is that all the sufficient statistics can be collected in a single scan of the data. Computation is linear in the number of observations and predictors, the lowest order possible if an algorithm is to inspect every data point. Secondly, missing predictors do not interfere with the estimation procedure or the prediction step. Even if all the X_j's are missing for a particular subject one can still offer the information available in $P(Y = 1)$. Naïve Bayes' indifference toward missing data is especially convenient in medical diagnosis where some indicants are unavailable, expensive, or unnecessary to produce an accurate diagnosis. This is not possible in logistic regression models without additional assumptions on the joint distribution of the predictors.

In spite of the often unrealistic conditional independence assumption, naïve Bayes still performs rather well, even when compared against classifiers that model the dependence structure. These studies have found naïve Bayes to be intuitive (Kononenko 1990), easily competitive with decision trees (Langley, Iba, and K.Thompson 1992), and no less competitive than their more complex counterpart, the Bayesian network (Friedman, Geiger, and Goldszmidt 1997). See Domingos and Pazzani (1997) for a more complete history of the naïve Bayes classifier and more reasons why it has maintained such a tradition in the machine learning community.

2.3 Boosting and the naïve Bayes classifier

As discussed in chapter 1, Freund and Schapire's *Discrete AdaBoost* algorithm proceeds as follows. Initialize the weights of each observation to $w_i^{(1)} = \frac{1}{N}$. For t in 1 to T iterate the following steps.

1. Learn model $H_t(x_i) : X \to [0, 1]$.

2. Compute $\epsilon_t = \sum_{i=1}^{N} w_i^{(i)} |y_i - H_t(x_i)|$ as the error for the model H_t.

3. Let $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.

4. Update the observation weights as $w_i^{(t+1)} = w_i^{(t)} \beta_t^{1-|y_i - H_t(x_i)|}$.

5. Normalize $w_i^{(t+1)}$ so that they sum to one.

The final boosted model suggested by Freund and Schapire that combines the models of each iteration is

$$H(x) = \frac{1}{1 + \prod_{t=1}^{T} \beta_t^{2r(x)-1}} \qquad \text{where} \qquad r(x) = \frac{\sum_{t=1}^{T} \left(\log \frac{1}{\beta_t}\right) H_t(x)}{\sum_{t=1}^{T} \log \frac{1}{\beta_t}}. \tag{2.5}$$

Boosting the naïve Bayes classifier involves substituting in the above algorithm $H_t = P_t(Y = 1|X)$. As described in section 2.2 estimation of $P_t(Y = 1|X)$ assumes that $P_t(Y = 1|X) \propto P_t(Y = 1)P_t(X_1|Y = 1) \cdots P_t(X_d|Y = 1)$.

The boosted naïve Bayes classifier has already seen some success. Charles Elkan's application of boosted naïve Bayes won first place out of 45 entries in the data mining competition KDD'97 (Elkan 1997). Elkan also showed that it has a non-linear logistic regression form and notes that it belongs to the computational complexity class NC. This class of algorithms meets the basic constraints of neurocomputational feasibility for low level brain processes, thus suggesting they are a plausible computational model for animal learning. However, while the usual non-boosted naïve Bayes approach leads to elegant and effective explanations (see, for example, Madigan, Mosurski, and Almond (1996) and Becker, Kohavi, and Sommerfield (1997)), the *AdaBoost*-ed version destroys this feature.

In what follows, I derive a "boosted weight of evidence" that the explanation procedures of Madigan, Mosurski, and Almond (1996) and Becker, Kohavi, and Sommerfield (1997) can use directly. I will express the *AdaBoost* voting scheme in a slightly different form and substitute a Taylor expansion for each of the boosted classifiers. In the end we will be able regain a boosted version that is additive as before.

Proposition 2.1 (Interpretable naïve Bayes) *The AdaBoost naïve Bayes classifier is approximately linear in the predictors on the logit scale. The approximation is* $\sum_t O\left((\log \frac{p_t}{1-p_t})^3\right)$ *where* p_t *is the* t^{th} *boosted classifier.*

Proof: Writing the *AdaBoost* combined classifier, $H(x)$, in the form of a log-odds simplifies the combination procedure.

$$\begin{aligned} \log \frac{H(x)}{1-H(x)} &= -\log \prod_{t=1}^{T} \beta_t^{2r(x)-1} \\ &= \sum_{t=1}^{T} (\log \beta_t)\left(1 - 2P_t(Y=1|X)\right) \end{aligned} \tag{2.6}$$

Note that

$$P_t(Y=1|X) = \frac{1}{1+\frac{P_t(Y=0|X)}{P_t(Y=1|X)}} = \left(1+e^{-\log \frac{P_t(Y=1|X)}{P_t(Y=0|X)}}\right)^{-1}. \tag{2.7}$$

Substituting equation 2.7 into equation 2.6 we obtain

$$\log \frac{H(x)}{1-H(x)} = \sum_{t=1}^{T} (\log \beta_t)\left(1 - 2\left(1+e^{-\log \frac{P_t(Y=1|X)}{P_t(Y=0|X)}}\right)^{-1}\right). \tag{2.8}$$

Substituting (up to the linear term) the Taylor approximation to the sigmoid function $\left(\frac{1}{1+e^{-x}} = \frac{1}{2} + \frac{1}{4}x + O(x^5)\right)$ produces a linear combination of the log-odds from each boosted naïve Bayes classifier:

$$\log \frac{H(x)}{1-H(x)} \approx \sum_{t=1}^{T} \left(\frac{1}{2}\log\frac{1}{\beta_t}\right) \log \frac{P_t(Y=1|X)}{P_t(Y=0|X)} \tag{2.9}$$

Each of the T classifiers are accurate to a first order log-odds term. Even though this might seem to be a poor approximation, it might do quite well given that we are interested in classification. The linear approximation to the sigmoid function is exact at 0 and loses precision as the argument moves away from 0. On the probability scale, the two functions agree at $P(Y=1|X) = \frac{1}{2}$. This is precisely the point where classification is the most in doubt. On the probability interval $\left[\frac{1}{4}, \frac{3}{4}\right]$ the absolute error of the approximation is less than 0.025. In the region near the extremes the approximation preserves the sign of the *AdaBoost* combined hypothesis but will differ on magnitude. Although the probability estimates will differ in this case, both methods will make the same classification. Note that this approximation applies to each of the probabilistic classifiers that compose the final model.

Finally, to help maintain some probabilistic interpretation of the final model (so that both sides of 2.9 appear as log-odds) and to remove some of the effect of the number of boosting iterations, we normalize the classifier weights $\left(\frac{1}{2}\log\frac{1}{\beta_t}\right)$. Intuitively, each probability model casts a log-odds vote for or against classification into $Y = 1$ where each models' vote is proportional to its quality. So letting $\alpha_t = \frac{\log\frac{1}{\beta_t}}{\sum_{t=1}^{T}\log\frac{1}{\beta_t}}$ and assuming a naïve Bayes model for $P_t(Y = 1|X)$, a boosted estimate for the log-odds in favor of $Y = 1$ is:

$$\sum_{t=1}^{T}\alpha_t \log\frac{P_t(Y=1)}{P_t(Y=0)} + \sum_{j=1}^{d}\sum_{t=1}^{T}\alpha_t \log\frac{P_t(X_j|Y=1)}{P_t(X_j|Y=0)}$$

$$\begin{aligned} &= \text{boosted prior weight of evidence} + \\ &\qquad \sum_{j=1}^{d} \text{boosted weight of evidence for } X_j \end{aligned} \qquad (2.10)$$

This again is just a naïve Bayes classifier where the estimates of the weights of evidence have been biased by boosting. The sigmoid function transforms the boosted log-odds to a boosted predicted probability as

$$P_t(Y = 1|X) = \frac{1}{1 + e^{-\text{log-odds}}}.$$

$\square$

Soon after the development of this version of the boosted naïve Bayes classifier Schapire and Singer produced the *Real AdaBoost* algorithm (Schapire and Singer 1998). This algorithm basically applies boosting on the logit-scale. Each classifier, $h_t(x)$, puts out a real value so that when $h_t(x)$ is negative we believe that $Y = 0$ and when $h_t(x)$ is positive we believe that $Y = 1$. The weights follow the update rule $w_i^{(t+1)} = w_i^{(t)} e^{-\alpha_t(2y_i-1)h_t(x_i)}$ where $\alpha_t \in (-\infty, \infty)$. The final prediction is $H(x) = \sum_t \alpha_t h_t(x)$.

Without approximation, boosting the naïve Bayes classifier via the Real *AdaBoost* algorithm also yields an interpretable version.

Proposition 2.2 *Applying Real AdaBoost to the naïve Bayes classifier yields a classifier that is linear in the predictors on the logit scale.*

Proof:

As in 2.3 let

$$\begin{aligned} h_t(x) &= \log \frac{P_t(Y=1|X)}{P_t(Y=0|X)} \\ &= \log \frac{P_t(Y=1)\prod_{j=1}^d P_t(X_j|Y=1)}{P_t(Y=0)\prod_{j=1}^d P_t(X_j|Y=0)} \\ &= w_0^{(t)} + \sum_{j=1}^d w_j^{(t)}(X_j) \end{aligned}$$

Since the final classifier of the Real *AdaBoost* is $H(x) = \sum_t \alpha_t h_t(x)$ we trivially obtain an interpretable classifier.

$$\begin{aligned} H(x) &= \sum_{t=1}^T \alpha_t w_0^{(t)} + \sum_{j=1}^d \sum_{t=1}^T \alpha_t w_j^{(t)}(X_j) \qquad (2.11) \\ &= \text{boosted prior weight of evidence} + \\ &\qquad \sum_{j=1}^d \text{boosted weight of evidence for } X_j \end{aligned}$$

If we let $\alpha_t = \frac{\log \frac{1}{\beta_t}}{\sum_{t=1}^T \log \frac{1}{\beta_t}}$ we will obtain the same expression as 2.10. Although the reweighting of the observations is slightly different, the final classifier has the same interpretable form.

□

Although conceptually it would seem as if fitting many models would lead to overfitting, boosting the naïve Bayes classifier does not seem to overfit. This may be due

Table 2.1: Datasets used for comparison

	N	Predictors	Positive cases
Knee diagnosis	99	26	44%
Diabetes	768	8	35%
Credit approval	690	15	44%
Coronary artery disease	303	13	44%
Breast tumors	699	9	34%

to its linearity (or near linearity in the case of *Discrete AdaBoost*) and relatively few effective parameters. The form of the final classifier contains only as many parameters as the original classifier with the addition of T, the number of rounds of boosting. Deciding when to stop boosting may require cross-validation.

2.4 Performance of the boosted naïve Bayes classifier

The substantial changes to the classifier combination step proposed in section 2.3 may have produced a voting method with different discrimination properties. In this section we show empirically on the five datasets described in table 2.4 that the misclassification rates for this new method are comparable to those generated by the *AdaBoost* algorithm. We note that every one of these datasets concerns a problem domain where human understanding of the machine reasoning process is likely to be important.

In a retrospective study, O'Kane, Ridgeway, and Madigan (1998) use the weight of evidence boosted naïve Bayes classifier to diagnose candidates for knee surgery at a sports medicine clinic. They present boosted parameter estimates and example evidence balance sheets that physicians can easily interpret and utilize in diagnosis. A prospective study is in progress. The remaining datasets are from the UCI repository

Table 2.2: Misclassification rates (standard deviation)

	Naïve Bayes	*Discrete AdaBoost*	Weight of evidence
Knee diagnosis	14.0% (5.0%)	13.8% (5.5%)	13.4% (5.7%)
Diabetes	25.0% (2.0%)	24.4% (2.5%)	24.4% (2.6%)
Credit approval	16.8% (2.0%)	15.5% (2.1%)	15.5% (2.1%)
Coronary artery disease	18.4% (3.0%)	18.3% (3.2%)	18.3% (3.3%)
Breast tumors	3.9% (1.0%)	3.8% (1.0%)	3.8% (1.0%)

(Merz and Murphy 1998). For each dataset we created 100 training sets randomly composed of two-thirds of the data. Then we compared the misclassification rate of naïve Bayes, *AdaBoost* naïve Bayes, and the proposed boosted weight of evidence naïve Bayes on the remaining one-third of the cases comprising the test set. Cross-validation on the training data yielded an estimate of the optimal number of boosting iterations. Table 2.2 compares the misclassification rates of the three methods. Both boosting methods demonstrate marginal absolute improvements in misclassification rate for most of the datasets. However, the most important result from table 2.2 is that boosted weight of evidence naïve Bayes almost perfectly matches the performance of the *AdaBoost* method. We have also examined aspects of the performance of the boosted weight of evidence naïve Bayes classifier that go beyond just misclassification rate. The so-called "mean Brier score" in equation 2.12 is a proper scoring rule that considers both discrimination ability and calibration:

$$\overline{B} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{P}(Y = 1|X))^2 \tag{2.12}$$

Yates (1982) and Spiegelhalter (1986) provide an extensive discussion of this scoring rule that has its roots in meteorology. Table 2.3 indicates that boosting generally decreases the mean Brier score and again the boosted weight of evidence naïve Bayes

Table 2.3: Mean Brier score (standard deviation)

	Naïve Bayes	*AdaBoost*	Weight of evidence
Knee diagnosis	0.107 (0.040)	0.108 (0.038)	0.103 (0.036)
Diabetes	0.170 (0.010)	0.164 (0.011)	0.164 (0.011)
Credit approval	0.127 (0.020)	0.112 (0.012)	0.113 (0.012)
Coronary artery disease	0.135 (0.020)	0.132 (0.016)	0.130 (0.021)
Breast tumors	0.035 (0.009)	0.035 (0.009)	0.034 (0.009)

classifier is competitive.

When a probabilistic prediction model assigns, e.g., a 30% probability to a positive outcome, one would like 30% of such cases to actually have a positive outcome. Various authors have proposed calibration scores and plots that attempt to represent this property. Calibration plots (Copas 1983a) show whether probabilistic predictions are calibrated. Figure 2.1 displays an example calibration plot for a random split of the data. The perfectly calibrated predictor will have a straight line. Across the five datasets, the plots generally show that the boosting methods provide improved calibration, but the two boosting methods are indistinguishable. Further research may show that boosting smooths predictions in a way that improves their calibration.

2.5 *Application to prediction of surgical intervention in patients with knee complaints*

In O'Kane, Ridgeway, and Madigan (1998) we applied the interpretable boosted naïve Bayes classifier to a problem in automated medical diagnosis. This section presents the findings of that joint work.

Knee complaints secondary to injury and overuse are common in both general and orthopedic practice. They are particularly common in athletes and other physically

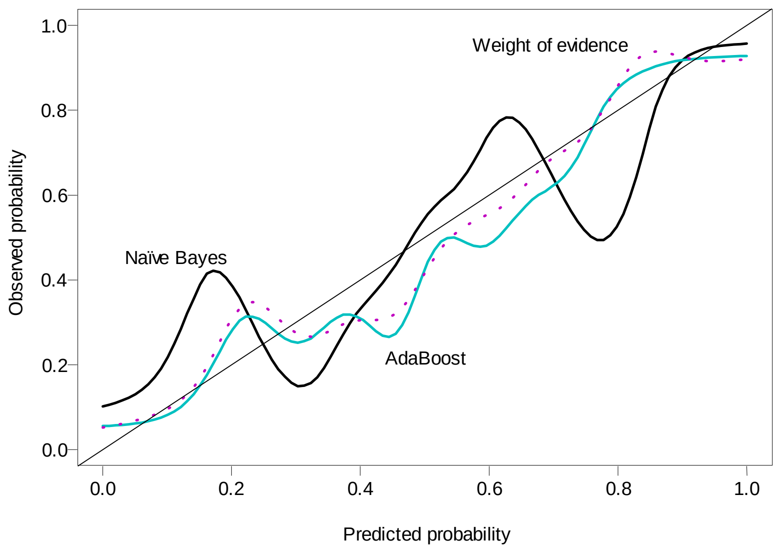

Figure 2.1: Example calibration plots for the credit dataset

active individuals. Knee problems including significant meniscal tears, anterior cruciate ligament (ACL) tears, intraarticular fractures and osteochondritis dessicans are often best managed with surgical treatment. On the other hand, problems including patellofemoral pain, medial collateral ligament sprains, iliotibial band syndrome, and patellar tendonitis are generally best managed non-operatively with appropriate rehabilitation. Based on history and physical exam, it can be difficult to separate those patients with knee pain likely to benefit from early surgical intervention from those in whom initial conservative treatment is more appropriate.

Findings are mixed in prior studies assessing the ability of clinical examination to predict the arthroscopic diagnosis in patients with knee complaints. Gibson, Davies, Crane, and Henry (1987) found that a clinical exam resulted in an unequivocal diagnosis of internal derangement in only 35% of the cases. They report a frequent discordance between clinical diagnosis and arthroscopic findings. Much of the literature focuses on the diagnostic accuracy of clinical exam to predict meniscal tears. While some authors have not found a clinical pattern that would reliably predict meniscal tears (Noble and Erat 1980), others have found that a combination of historical and physical examination variables can predict meniscal tears with some accuracy (Barry, Smith, McManus, and MacAuley 1983). Anderson and Lipscomb (1986) found at least one positive mechanical test in 79% of meniscal tears. Alternatively, Curtin, O'Farrell, McGoldrick, Dolan, Mullan, and Walsh (1992) found that for 175 patients taken to arthroscopy, clinical exam and plain radiography demonstrated poor specificity for medial meniscal tears and poor sensitivity for lateral meniscal tears. They demonstrated better specificity for ACL tears and of 30 predicted, 26 were confirmed. There were 7 ACL tears discovered arthroscopically that were not diagnosed on clinical exam. Finally, Abdon, Lindstrand, and Thorngren (1990) found that while clinical accuracy in detecting meniscal tears was 61%, employing a multivariate analysis of 68 different clinical variables could correctly predict a meniscal tear in 80% of the cases. This last study, like the others, demonstrates that making an accurate diagnosis based on history and physical is difficult. It suggests though, that analysis of the variables statistically may be more accurate than the clinical impression based on those variables.

While the literature cited examines the accuracy of diagnosis following history and clinical exam, an important question not directly addressed is the likelihood that a patient presenting with a knee complaint will benefit from surgical intervention. For the primary care physician contemplating a referral for surgery, or for a surgeon contemplating arthroscopy, the likelihood the patient will benefit from the procedure is of primary concern. This application attempts to determine whether or not historical and clinical variables at the time of presentation can accurately predict if a patient is likely to have a knee problem requiring knee surgery. It also examines the role of the boosted naïve Bayes classifier to more accurately predict the answer to this clinical question.

Methods

Data were collected through a retrospective chart review in a university based orthopedic sports medicine clinic. Charts were pulled sequentially in alphabetical order and the record was reviewed for all knee diagnoses. Data were collected for all patients in whom the surgical or non-surgical treatment was satisfactorily completed. The patients' age and gender were noted and binary data were collected for the historical and clinical variables.

We evaluated our model according to procedures common in the statistics and machine learning communities. We first randomly divided the patient sample of 99 observations into two groups, a training set and a test set. We estimated the parameters of the boosted naïve Bayes model as well as the optimal number of boosting iterations using only the training set. The estimated model was then used to predict the necessity of knee surgery on the patients in the test set and compute the misclassification rate of the test set. Repeating these steps for several partitions of the sample and averaging the misclassification rate for each partition yields an estimate of the accuracy of the classifier on future observations. We also varied the proportion of the observations used in the training set to estimate a "learning curve" for the knee injury classification problem that helps determine the number of patients needed for constructing an accurate model.

Table 2.4 and table 2.5 show the estimated weights of evidence, $\hat{w}_j(X_j)$. The point estimates shown are the expected value of the boosted weight of evidence. When estimating the probabilities in equation 2.10 we take a Bayesian approach and assume that each probability has a beta distribution. With a Beta(1,1) prior on each parameter we obtain numerically stable estimates, even for classes with low cell counts. This is equivalent to what is known as the Laplace correction. Let p_0 represent $P(X_j = x_j|Y = 0)$ and p_1 represent $P(X_j = x_j|Y = 1)$ for some arbitrary j. For point estimates of the weights of evidence we would like to compute

$$\mathrm{E}\ w_j(x_j) = \mathrm{E}\left(\log\frac{p_1}{p_0}\right) = \mathrm{E}\log p_1 - \mathrm{E}\log p_0 \tag{2.13}$$

where p_0 and p_1 have some posterior distribution Beta(α_0, β_0) and Beta(α_1, β_1) re-

Table 2.4: Estimated weights of evidence – 1 of 2

Prior		-1 (11)
Variable	**negative**	**positive**
Unilateral	-29 (64)	5 (7)
Injury	**-50 (23)**	**39 (14)**
Locking	**-6 (3)**	**172 (50)**
Instability	**-14 (5)**	**88 (50)**
Mechanical	-1 (31)	-4 (25)
Anterior pain	0 (17)	-4 (38)
Local pain	23 (32)	-10 (12)
Sports related	14 (15)	-19 (15)
Industrial	-5 (8)	34 (96)
Prior surgery	12 (8)	-26 (41)
Prior injury	-7 (9)	20 (33)
Depression	-5 (8)	14 (64)
Effusion	**-72 (24)**	**85 (29)**
Range loss	-8 (15)	12 (22)
Instability MCL	6 (4)	**-140 (53)**
Instability LCL	**-4 (2)**	**133 (36)**
Instability ACL	**-34 (7)**	**298 (19)**
Patella crepitus	-11 (29)	13 (34)
McMurray's	**-38 (25)**	56 (52)
Tender med JL	**-30 (14)**	**49 (18)**
Tender lat JL	**-20 (9)**	**50 (29)**
Tender anterior	**13 (6)**	**-72 (89)**

Table 2.5: Estimated weights of evidence – 2 of 2

	Male	Female
Sex	-6 (13)	8 (14)

	≤ 24	25-32	33-46	≥ 47
Age	-3 (37)	28 (22)	-1 (26)	-12 (23)

	None	Slow	Rapid
Swelling	-41 (40)	58 (37)	-29 (23)

	No	Yes	Arthritis
XR-fracture	64 (75)	-22 (14)	45 (60)

spectively. I will use the fact that

$$\begin{aligned} \mathrm{E} \log p &= \int_0^1 \log(p) \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} p^{\alpha-1}(1-p)^{\beta-1} dp \\ &= \psi^{(0)}(\alpha) - \psi^{(0)}(\alpha+\beta) \end{aligned}$$

where $\psi^{(n)}(z)$ is the digamma function, the $(n+1)^{th}$ derivative of the logarithm of the gamma function. So the expectation in 2.13 is

$$\mathrm{E}\ w_j(x_j) = \psi^{(0)}(\alpha_1) - \psi^{(0)}(\alpha_0) - \psi^{(0)}(\beta_1) + \psi^{(0)}(\beta_0) \tag{2.14}$$

This expectation also holds for the prior weight of evidence. An estimate of the boosted weights of evidence is simply a weighted average of 2.14. The variance of the weights is not so simple. The boosting iterations are highly dependent on one another. We appeal to bootstrapping to estimate the variance of the w_j's.

Returning to tables 2.4 and 2.5, the number shown in parentheses is a bootstrap estimate of the standard deviation of $\hat{w}_j(X_j)$. If the weight of evidence estimates were normally distributed then the ratio of the estimate to the estimated standard

Table 2.6: Example evidence balance sheet. Weights are scaled by 100.

Evidence in favor of knee surgery		Evidence against knee surgery	
Female	+8	Prior	-1
Knee is unstable	+88	Age 50	-12
Knee locks	+172	No effusion	-72
Tender med JL	+49	Negative McMurray's	-38
Total positive evidence	+317	Total negative evidence	-123
Total evidence		+194	
Probability of knee surgery		87%	

deviation would provide a t-test statistic for testing whether the weight of evidence differed significantly from 0. However, the bootstrap distribution of the estimates were often skewed, indicating substantial departures from normality. According to the bootstrap distribution, if $\hat{P}(\hat{w}_j(X_j) > 0)$ is less than 0.025 or exceeds 0.975 (evidence that the weight of evidence is strongly negative or strongly positive respectively) then we boldfaced the variable in tables 2.4 and 2.5. This amounts to a $\alpha = 0.05$ test based on the bootstrap percentile interval (Efron and Tibshirani 1993). This is analogous to the $p < .05$ used to determine statistical significance in other statistical models. The bold faced weights of evidence indicate that the associated variable is a "significant" predictor for or against surgery.

The weights in tables 2.4 and 2.5 are additive. That is, to determine the total weight of evidence in favor of a patient needing knee surgery the physician merely needs to elicit a collection of indicants, sum the associated weights, and note the magnitude and sign of the total weight.

For a new patient a physician could easily create an evidence balance sheet

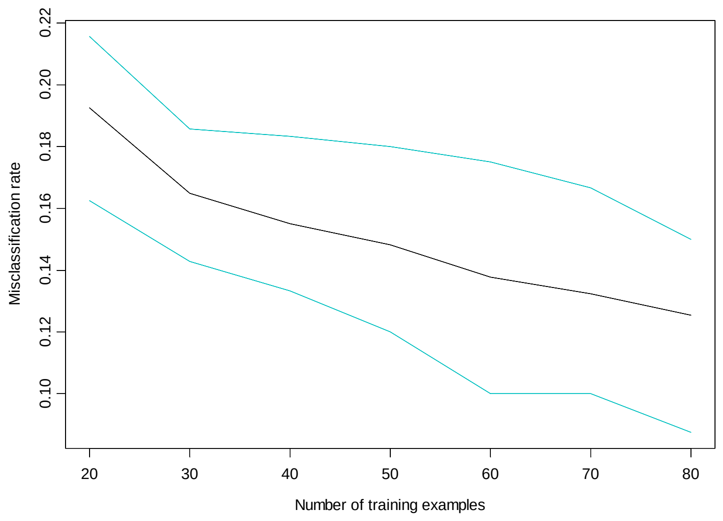

Figure 2.2: Learning curve for predicting the necessity of knee surgery (mean misclassification with 1^{st} and 3^{rd} quartiles)

(Spiegelhalter and Knill-Jones 1984) from tables 2.4 and 2.5. Table 2.6 shows an example of such an evidence balance sheet where the weights of evidence are scaled by a factor of 100 for readability. Figure 2.2 shows the estimated learning curve for the necessity of knee surgery. Training on 66 patients we expect to predict knee surgery with 87% accuracy (misclassification rate is approximately 13%).

2.6 Conclusion

In this chapter I proposed a method for merging the interpretation properties of the naïve Bayes classifier with the performance gains associated with boosting. In particular, via an intuitively reasonable approximation, I showed that the boosted naïve Bayes classifier is nearly linear in the predictors. Through empirical study on five datasets, this method demonstrated slight improvements over the usual naïve Bayes in terms of misclassification rates. Probably of more interest, as far as this method is concerned, is that both boosting methods had near identical performance.

Although misclassification seems to be the error metric of choice in the machine learning community it need not be the only one. On metrics that measure calibration such as the Brier score decomposition or simple calibration plots both boosting methods out performed the exaggerated naïve Bayes prediction.

For applications to large datasets, the naïve Bayes classifier alone is viable. For discrete predictors, a linear scan of the dataset provides all the sufficient statistics for parameter estimation. If multiple scans are allowable boosting has an extra cost that is linear in the number of boosting iterations. For the cost of an extra scan, the classifier might improve calibration.

Chapter 3

BOOSTED NAÏVE BAYES REGRESSION

This chapter proposes a method for applying the boosted naïve Bayes classifier to regression problems. The paper Ridgeway, Madigan, and Richardson (1999) contains a condensed version of these ideas.

Classification problems have dominated research on boosting to date. The application of boosting to regression problems, on the other hand, has received little investigation. In this paper we develop a new boosting method for regression problems. We cast the regression problem as a classification problem and apply an interpretable form of the boosted naïve Bayes classifier. This induces a regression model that we show to be expressible as an additive model for which we derive estimators and discuss computational issues. We compare the performance of our boosted naïve Bayes regression model with other interpretable multivariate regression procedures.

3.1 Introduction

In a wide variety of classification problems, boosting techniques have proven to be an effective method for reducing bias and variance, and improving misclassification rates (Bauer and Kohavi 1999). While more evidence compiles about the utility of these techniques in classification problems, little is known about their effectiveness in regression problems. Freund and Schapire (1997) provide a suggestion as to how boosting might produce regression models using their algorithm *AdaBoost.R*. Breiman (1997) also suggests how boosting might apply to regression problems using his algorithm *arc-gv* and promises a study in the near future. The only actual implementation and experimentation with boosting regression models that we know of is Drucker (1997) in which he applies an *ad hoc* modification of *AdaBoost.R* to some regression problems and obtains promising results.

In this chapter we develop a new boosting method for regression problems. Motivated by the concept behind *AdaBoost.R*, we project the regression problem into a classification problem on a dataset of infinite size. We use a variant of the boosted naïve Bayes classifier (Ridgeway, Madigan, Richardson, and O'Kane 1998) that offers flexibility in modeling, predictive strength, and, unlike most voting methods, interpretability. In spite of the infinite dataset we can still obtain closed form expressions for the parameter estimates within each iteration of the boosting algorithm. As a consequence of the model formulation, the naïve Bayes regression model turns out to be an estimation procedure for additive regression for a monotone transformation of the response variable. In this paper we derive the boosted naïve Bayes regression model (BNB.R) as well as show some results from experiments using a discrete approximation.

3.2 Boosting for classification

In binary classification problems we observe $(\underline{X}, Y)_i$, $i = 1, \cdots, N$ where $Y \in \{0, 1\}$ and we wish to form a model, $h(X)$, which accurately predicts Y. Boosting describes a general voting method for constructing $h(X)$ from a sequence, $h_t(X)$, where each model uses a different weighting on the observations in the dataset to estimate its parameters. Observations poorly modeled by h_t receive greater weight for learning h_{t+1}. The final boosted model is a combination of the predictions from each h_t where each is weighted according to the quality of its classification on the training dataset. Freund and Schapire (1997) presented a boosting algorithm for classification problems that empirically has yielded reduction in bias, variance, and misclassification rates with a variety of base classifiers and problem settings. Variants of their *AdaBoost* (adaptive boosting) algorithm have become the dominant form of boosting in practice and experimentation so far.

As discussed in chapter 1, Freund and Schapire's *Discrete AdaBoost* algorithm proceeds as follows. Initialize the weights of each observation to $w_i^{(1)} = \frac{1}{N}$. For t in 1 to T iterate the following steps.

1. Learn model $H_t(x_i) : X \to [0, 1]$.

2. Compute $\epsilon_t = \sum_{i=1}^{N} w_i^{(i)} |y_i - H_t(x_i)|$ as the error for the model H_t.

3. Let $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.

4. Update the observation weights as $w_i^{(t+1)} = w_i^{(t)} \beta_t^{1-|y_i - H_t(x_i)|}$.

5. Normalize w_i^{t+1} so that they sum to one.

The final boosted model suggested by Freund and Schapire that combines the models of each iteration is

$$H(x) = \frac{1}{1 + \prod_{t=1}^{T} \beta_t^{2r(x)-1}} \qquad \text{where} \qquad r(x) = \frac{\sum_{t=1}^{T} \left(\log \frac{1}{\beta_t}\right) H_t(x)}{\sum_{t=1}^{T} \log \frac{1}{\beta_t}} \tag{3.1}$$

They prove that boosting in this manner places an upper bound on the final misclassification rate on the training dataset at

$$2^{T-1} \prod_{t=1}^{T} \sqrt{\epsilon_t (1 - \epsilon_t)}. \tag{3.2}$$

Note that as long as the weighted misclassification rate of each of the classifiers can do even slightly better (or worse) than random guessing, then the bound decreases. In 3.1 when computing $r(x)$, the weight coefficient for a particular classifier is $\log \frac{1}{\beta_t}$. This coefficient is 0 when the classifier is no better than random guessing, $\epsilon_t = \frac{1}{2}$. Also, if the classifier does worse than random guessing, $\epsilon_t > \frac{1}{2}$, then $\beta_t > 1$ and so the weight coefficient is negative, effectively reversing the predictions made by $H_t(x)$.

Even if boosting drives the training error to zero the boosted models tend not to be overfit on real datasets. The work on *AdaBoost* also produced bounds on generalization error based on VC dimension. However, *AdaBoost*'s performance in practice often is much better than the bound implies. Empirical evidence has shown that the base classifier can be fairly simplistic (shallow classification trees) and yet, when boosted, can capture complex decision boundaries (Breiman 1998). Ridgeway, Madigan, Richardson, and O'Kane (1998) substituted the naïve Bayes classifier $h_t(X)$ and a Taylor expansion for the sigmoid function to obtain an accurate and interpretable boosted naïve Bayes classifier. Using the same boosting algorithm presented above, the step combining the naïve Bayes classifiers is approximately

$$\begin{aligned}\sum_{t=1}^{T}\alpha_t \log\frac{P_t(Y=1)}{P_t(Y=0)} + \sum_{j=1}^{d}\sum_{t=1}^{T}\alpha_t \log\frac{P_t(X_j|Y=1)}{P_t(X_j|Y=0)} \\ = \text{ boosted prior weight of evidence} + \\ \sum_{j=1}^{d}\text{boosted weight of evidence for } X_j \qquad (3.3)\end{aligned}$$

where $\alpha_t = \frac{\beta_t}{\sum_t \beta_t}$. $P_t(\cdot)$ is an estimate of the probability density function using a weighted likelihood taking into account the observation weights, $w^{(t)}$, from the t^{th} boosting iteration. The boosting weights of evidence are a version of those described in Good (1965). A positive weight corresponding to X_j indicates that the state of X_j is evidence in favor of the hypothesis that $Y = 1$. A negative weight is evidence for $Y = 0$.

In practice the non-boosted naïve Bayes classifier consistently demonstrates robustness to violations in its assumptions and tends not to be sensitive to extraneous predictor variables. Note that 3.3 remains a naïve Bayes classifier even though it has been boosted. However, boosting has biased the estimates of the weights of evidence to favor improved misclassification rates. Subsequent classifiers place more weight on observations that are poorly predicted. Intuitively, boosting weights those regions of the sample space that are not modeled well or exemplify violations of the model's assumptions (in the naïve Bayes case, conditional independence of features given the unknown class). Similar to the weights of evidence logistic regression proposal of Spiegelhalter and Knill-Jones (1984), boosting the naïve Bayes classifier seems to have a shrinking effect on the weights of evidence and reins in the classifiers exaggerated probability predictions leading to improved calibration.

Efforts to improve upon the naïve Bayes classifier by relaxing the naïve conditional independence assumption are not always successful. Friedman and Goldszmidt (1996) compared the performance of the naïve Bayes classifier and, their more powerful counterpart, the Bayesian network. They found that the complexity in fitting the Bayesian networks models did not offset the generally small gains in predictive performance. However, in response, they proposed the augmented naïve Bayes that allows for some dependence among the features. In particular they develop tree-augmented naïve Bayes (TAN) and show that it outperforms naïve Bayes as well as *C4.5*, a decision tree classifier. Keogh and Pazzani (1999) confirmed that TAN outperforms naïve Bayes on average, at times substantially but more often slightly. Building on the work of Friedman and Goldszmidt (1996) they developed an efficient hill-climbing search (HCS) for an augmented naïve Bayes classifier. On the thirteen UCI datasets analyzed, HCS outperforms TAN, decreasing misclassification rates up to 10%.

Other methods to offset violations of the naïve Bayes assumption build decision trees that fit local naïve Bayes classifiers at the leaves. Zheng and Webb (1998) give a history of some methods as well as propose a new method of their own. Within a leaf this method fits a naïve Bayes classifier where the observations weighting assigns weight 1 to observations in the leaf and weight 0 to observations outside the leaf. The final model then mixes all the leaves together. Boosting performs in a similar manner. However, rather than partitioning the dataset, boosting reweights smoothly, learning on each iteration to what degree it should fit the next classifier to each observation.

Efforts to use the naïve Bayes assumption in regression applications has failed. Frank, Trigg, Holmes, and Witten (1998) showed that direct application of the naïve

Bayes assumption performs worse than linear regression and regression trees. The robust properties of naïve Bayes may only exist for classification. In this work we will be able to take advantage of these robust classification properties and apply them to regression problems to outperform linear regression and trees in some settings.

3.3 Boosting regression problems

In spite of the attention boosting receives in classification methodology, few results exist that apply the ideas to regression problems. If boosting's effectiveness extends beyond classification problems then we might expect that the boosting of simplistic regression models could result in a richer class of regression models. Breiman (1997) describes a boosting method called *arc-gv* although to date no performance results have been published.

Drucker (1997) considered an *ad hoc* boosting regression algorithm. He assigned a weight, w_i, to each observation and fit a CART model, $h(x) \to Y$, to the weighted sample. Similar to the *AdaBoost* algorithm for classification he set

$$\epsilon_t = \sum_{i=1}^{N} w_i^{(t)} L\left(\frac{|y_i - h_t(x_i)|}{\max|y_i - h_t(x_i)|}\right) \tag{3.4}$$

He offers three candidate loss functions for L, all of which are constrained to the [0,1] interval. The definition of β_t remains the same and the reweighting proceeds in *AdaBoost* fashion.

$$w_i^{(t+1)} = w_i^{(t)} \beta_t^{1-L\left(\frac{|y_i - h_t(x_i)|}{\max|y_i - h_t(x_i)|}\right)} \tag{3.5}$$

In this manner, each boosting iteration constructs a regression tree on a different weighting of the dataset. Lastly, he suggests using a weighted median to merge the predictions of each regression tree. Using this method, his empirical analysis showed consistent improvement in prediction error over non-boosted regression trees.

Drucker's and Freund and Schapire's methods share little in common. In order to extend their theoretical classification results to regression problems Freund and

Table 3.1: Example dataset D

X_1	X_2	Y
0.6	0.4	0.3
0.8	0.5	0.9

Schapire project the regression dataset into a classification dataset and apply their *AdaBoost* algorithm. Our algorithm proceeds similarly.

3.3.1 Projecting the observed data

Freund and Schapire project the data into a "reduced *AdaBoost* space," a classification dataset in the following way. For the moment we will assume that $Y \in [0, 1]$ but the methodology readily extends to the whole real line. To make this transition to a classification problem we first expand the size of the dataset. Consider the toy dataset shown in table 3.1. We transform the original regression dataset, D, to a new classification dataset, D^*, as follows.

First, let S be a sequence of m equally spaced values in the interval [0,1]. Secondly, form the Cartesian product of (X_1, X_2, Y) and S. Then append the dataset with a binary variable, Y^*, that has the value 0 if $S < Y$ and 1 if $S \geq Y$. Table 3.2 shows the transformation of table 3.1. We will call this dataset D^* which has $m \times N$ observations. Now we can construct a classifier of the form $h : (S, X) \to \{0, 1\}$. In other words, we can give this model an X and an S and ask of it whether the Y associated with X is larger or smaller than S. A probabilistic classifier may instead give a probabilistic prediction so that $h : (S, X) \to [0, 1]$. Note that when m is large enough such that the precision of S exceeds the precision of Y the transformation of D to D^* is 1-to-1. and, therefore, the classification dataset contains the same information as the regression dataset. Throughout this chapter we will index the observations in D by i and the observations in D^* by (i, S).

Table 3.2: Transformation of example dataset D

X_1	X_2	Y	S	$Y^* = I(S \geq Y)$
0.6	0.4	0.3	0.00	0
0.6	0.4	0.3	0.01	0
0.6	0.4	0.3	$\vdots$	0
0.6	0.4	0.3	0.29	0
0.6	0.4	0.3	0.30	1
0.6	0.4	0.3	0.31	1
0.6	0.4	0.3	$\vdots$	1
0.6	0.4	0.3	0.99	1
0.6	0.4	0.3	1.00	1
0.8	0.5	0.9	0.00	0
0.8	0.5	0.9	0.01	0
0.8	0.5	0.9	$\vdots$	0
0.8	0.5	0.9	0.89	0
0.8	0.5	0.9	0.90	1
0.8	0.5	0.9	0.91	1
0.8	0.5	0.9	$\vdots$	1
0.8	0.5	0.9	0.99	1
0.8	0.5	0.9	1.00	1

At this point our methodology and Freund and Schapire's methodology depart. Using their *AdaBoost.R* one fits a regression model on the regression dataset, D, which in turn induces a classifier on the classification dataset, D^*. That is, one can ask of the regression model whether it predicts the Y to be greater than or less than a value S given a vector of features X. The performance of this induced classifier on D^* determines the reweighting of the observations and the weight of each of the h_t's.

However, both Freund and Schapire's and Drucker's algorithms halt if the misclassification rate of any individual h_t on D^* exceeds $\frac{1}{2}$. Their algorithms iterate until $\epsilon_t > \frac{1}{2}$. In practice no method can guarantee that this constraint should hold long enough to build a suitable model. We noted earlier that the voting scheme of the original *AdaBoost* algorithm for binary classification problems 3.1 simply reverses the polarity of any classifier for which $\epsilon_t > \frac{1}{2}$. In terms of the voting scheme, these classifiers are just as useful as the ones for which $\epsilon_t < \frac{1}{2}$. This observation led us to investigate whether we could avoid fitting a regression model that induces a classifier and instead fit a classifier directly to D^*. In binary classification problems, if a classifier performs very poorly in the sense of getting almost every observation wrong, *AdaBoost* can use such a classifier just as much as one that gets almost everyone right.

3.4 Classification for infinite datasets

If $h(X, S)$ is our classifier for D^*, our predicted value of Y for a given X is the smallest value of S for which h predicts $Y^* = 1$. Many classifiers base their classification rule on estimates of $P(Y^* = 1|X, S)$. Therefore, to obtain a prediction for Y we can use

$$\hat{Y} = \inf_s \left\{ s : P(Y^* = 1|X, S) \geq \frac{1}{2} \right\}. \tag{3.6}$$

More easily stated, this prediction is the y for which we are equally uncertain whether the true Y is smaller or larger. Concretely, if $P(Y^* = 1|X, S = 0.3) = 0.1$ then we would believe that $Y^* = 0$ is more likely and therefore, by the definition of Y^*, Y is more likely to be larger than 0.3. On the other hand, if $P(Y^* = 1|X, S = 0.3) = 0.5$

then our beliefs would be divided as to whether Y is larger or smaller than 0.3. In this situation, 0.3 would make a reasonable prediction for Y.

3.4.1 Boosted naïve Bayes classification for infinite datasets

Generally naïve Bayes classification assumes that the features are independent given the class label. In the setting here the features consist of X and S and the class label, Y^*. This model corresponds to the following factorization.

$$P(Y^* = y^*|X_1, \cdots, X_d, S) \propto P(Y^* = y^*)P(S|Y^* = y^*) \prod_{j=1}^{d} P(X_j|Y^* = y^*) \tag{3.7}$$

This conditional independence assumption is not necessarily sensible. If in fact Y and X are positively correlated then, given $Y^* = 1$, knowledge that S is small is highly informative that Y is small and so X is likely to be small. Therefore, on the surface the naïve Bayes assumption does not seem reasonable. We then must rely on its robustness toward such violations and boosting's ability to compensate for incorrectly specified models. Indeed we will see that this in fact happens. As we will see here, assuming this factorization implies that the naïve Bayes regression model fits an additive model to a transformation of the response variable.

Proposition 3.1 (Naïve Bayes regression fits an additive model) *If $P(S = s|Y^*)$ is a continuous function of s and the factorization in equation 3.7 holds, then the prediction rule in equation 3.6 is an additive model for a transformation of the response variable Y.*

Proof:

Note that for equation 3.7 there exists s_1 and s_2 for every X such that

$$P(Y^* = 1|X, S = s_1) < \frac{1}{2} \text{ and } P(Y^* = 1|X, S = s_2) \geq \frac{1}{2}. \tag{3.8}$$

This follows by the construction of S

$$\lim_{S\to-\infty} P(S|Y^* = 1) = 0 \text{ and } \lim_{S\to\infty} P(S|Y^* = 0) = 0 \tag{3.9}$$

This implies that

$$\lim_{S\to-\infty} P(Y^*=1|X,S)=0 \text{ and } \lim_{S\to\infty} P(Y^*=1|X,S)=1 \tag{3.10}$$

Therefore, for the naïve Bayes model, 3.8 holds for some s_1 and s_2.

Substituting 3.7 into 3.6 the computation of the regression prediction under this model becomes

$$\hat{Y}=\inf_s \left\{ s : \log\frac{P(s|Y^*=0)}{P(s|Y^*=1)} \le \log\frac{P(Y^*=1)}{P(Y^*=0)} + \sum_{j=1}^{d}\log\frac{P(X_j|Y^*=1)}{P(X_j|Y^*=0)} \right\} \tag{3.11}$$

Note that equation 3.11 bears some resemblance to equation 3.3. We will call the function to the left of the inequality $\ell(s)$. $\ell(s)$ is necessarily non-increasing since as s increases it must become more likely that $Y^*=1$. Then, large values on the right side are evidence in favor of $Y^*=1$. Since 3.8 is true for the naïve Bayes model, if $\ell(s)$ is a continuous function of s (as would be the case for a smooth density estimator), then by the intermediate value theorem there exists some value of s for which the equality holds. In this case, 3.11 simplifies as

$$\log\frac{P_{S|Y^*=0}(\hat{Y}|Y^*=0)}{P_{S|Y^*=1}(\hat{Y}|Y^*=1)} = \log\frac{P(Y^*=1)}{P(Y^*=0)} + \sum_{j=1}^{d}\log\frac{P(X_j|Y^*=1)}{P(X_j|Y^*=0)} \tag{3.12}$$

or $\ell(\hat{Y})=f_0+\sum_{j=1}^{d} f_j(X_j)$.

Thus, if $\ell(s)$ is continuous, the naïve Bayes regression model is an additive model (Hastie and Tibshirani 1990) for a transformation of the response.

□

Estimation of the additive regression model shown in 3.12 is not traditional since the model relies on probability estimates rather than on backfitting (Friedman and Stuetzle 1981). Also, in the usual additive model framework, transformations of the response usually take the form of a transformation that stabilizes the variance (AVAS). Here, fitting a transformation of the response is a component of the model estimation procedure. Some earlier work on boosted naïve Bayes classification (Elkan

1997) showed that it was equivalent to a non-linear form of logistic regression. Recent work by (Friedman, Hastie, and Tibshirani 1998) shows that boosting fits an additive logistic regression model with a modified fitting procedure.

3.5 Parameter estimation

The model components of the usual (non-boosted) naïve Bayes classifier, consisting of $P(Y^*=1)$, $P(S|Y^*=0)$, $P(S|Y^*=1)$, $P(X_j|Y^*=0)$, and $P(X_j|Y^*=1)$, are fairly straightforward to estimate. The classification dataset, D^*, has $m \times N$ observations. If $Y \in [0,1]$ then we let m be large enough so that the precision of S exceeds the precision of Y and fit the usual naïve Bayes classifier. For example, an mle for $P(Y^*=1)$ is simply the count of rows for which $Y^*=1$ divided by $m \times N$. Estimators for $P(X_j|Y^*)$ when X_j is discrete is also a simple ratio of counts. Estimation of $P(S|Y^*)$ and $P(X_j|Y^*)$ when X_j is continuous might require a density estimate or discretization.

Clearly, managing such a dataset or restricting this method to $Y \in [0,1]$ limits its applicability. First I will show that it is not necessary to actually create D^* and then deal with the [0,1] restriction.

Estimation remains tractable as $m \to \infty$ and the resolution of S and Y^* become more refined. To demonstrate this consider the simplest part of the problem, that of estimating $P(Y^*=1)$, as $m \to \infty$. Let $S_j = \frac{j-1}{m-1}$, $j = 1, \cdots, m$, and $\lceil \cdot \rceil$ indicates the greatest integer function.

$$\begin{aligned} \hat{P}(Y^*=1) &= \lim_{m\to\infty} \frac{1}{m \times N} \sum_{i=1}^{N} \sum_{j=1}^{m} I(Y_i^*(S_j)=1) \\ &= \frac{1}{N} \sum_{i=1}^{N} \lim_{m\to\infty} \frac{1}{m} \sum_{j=1}^{m} I(S_j > y_i) \\ &= \frac{1}{N} \sum_{i=1}^{N} \lim_{m\to\infty} \frac{1}{m} \lceil (m-1)(1-y_i) \rceil \\ &= 1 - \bar{y} \end{aligned} \tag{3.13}$$

This says that if we randomly select an observation, i, from D and draw a number, S, from a Uniform[0,1], $\hat{P}(Y_i^*(S)=1)=1-\bar{y}$. In the presence of sufficient data we would believe that this would be close to $P(S>Y)$ if Y was a new observation drawn from the same distribution as the observations comprising D. Some difficulties arise, however, even in this simplest component of the model, when we consider $Y \in \Re$. In particular, the S_j's are not definable. Clearly we cannot generate m equally spaced S_j's on $(-\infty,\infty)$. To accommodate this we assign a finitely integrable weight function to each observation, $w_i(s)$, presumably with most of the mass in the neighborhood of y_i. We constrain these weight functions so that

$$w_i(s) \geq 0 \text{ and } \sum_{i=1}^{N} \int_{-\infty}^{\infty} w_i(s)ds = 1 \tag{3.14}$$

and, at least initially, we will fix $\int_{-\infty}^{\infty} w_i(s)ds = \frac{1}{N}$. Without these weight functions, the classifier estimation "tried" equally hard to correctly classify each observation in D^*. Since we only want to use this classifier for regression prediction we are really only interested in its performance in the neighborhood of the y_i's. To accomplish this we select $w_i(s)$ so that most of the mass rests near y_i and fit the classifier using these weight functions.

Fitting the classifier incorporating these weight functions introduces the need for product integration (Dollard and Friedman 1979). Product integration, $\prod$, extends the product, $\prod$, as integration, $\int$, extends the sum, $\sum$. The values of (i,s) index the observations in D^*. We can then compose the likelihood function as a product with respect to the discrete i and a product integral with respect to the continuous s over the joint density of each of the observations in D^*.

$$L(\theta) = \prod_{i=1}^{N} \prod_{-\infty}^{\infty} f(y_i^*(s), s, x_i|\theta)^{Nw_i(s)ds} \tag{3.15}$$

Here θ represents the collection of parameters that we need to estimate. Note that at this point the naïve Bayes assumption has not entered, but we have made another non-trivial assumption. To write the likelihood as in 3.15 we have assumed that the observations in D^* are mutually independent. This is certainly not true but we will see that with boosting it performs well nonetheless.

3.5.1 Derivation of the parameters

Proceeding then, the log-likelihood is

$$\ell(\theta) = \sum_{i=1}^{N} \int_{-\infty}^{\infty} N w_i(s) \log f(y_i^*(s), s, x_i|\theta) ds. \tag{3.16}$$

The naïve Bayes factorization assumes that $s \perp\!\!\!\perp x_1 \perp\!\!\!\perp \cdots \perp\!\!\!\perp x_n | y_i^*(s)$ and so

$$\ell(\theta) = \sum_{i=1}^{N} \int_{-\infty}^{\infty} N w_i(s) \log \left(f(y_i^*(s)|\theta) f(s|y_i^*(s), \theta) f(x_i|y_i^*(s), \theta) \right) ds.$$

The parameter vector θ contains the components of the naïve Bayes regression model, $\theta = \{P(y^* = 1), f(s|y^*), P(x = x'|y^*)\}$. The last term is $f(x|y^*)$ if x is continuous. We can use maximum likelihood to obtain estimates of these three parameters, which for the moment we will label p, $f(s|0)$, $f(s|1)$, $f(x|0)$ and $f(x|1)$. The derivation will begin with a maximum likelihood estimator for p.

$$\begin{aligned}
\frac{\partial \ell(\theta)}{\partial p} &= \sum_{i=1}^{N} \int_{-\infty}^{\infty} N w_i(s) \frac{d}{dp} \log \left(p^{y_i^*(s)} (1-p)^{1-y_i^*(s)} \right) ds \\
&= \sum_{i=1}^{N} \int_{-\infty}^{\infty} N w_i(s) \left[\frac{y_i^*(s)}{p} - \frac{1-y_i^*(s)}{1-p} \right] ds \\
&= \sum_{i=1}^{N} \int_{-\infty}^{y_i} N w_i(s) \left[\frac{0}{p} - \frac{1-0}{1-p} \right] ds + \sum_{i=1}^{N} \int_{y_i}^{\infty} N w_i(s) \left[\frac{1}{p} - \frac{1-1}{1-p} \right] ds
\end{aligned}$$

By definition, $y_i^*(s)$ is identically 0 on $(-\infty, y_i)$ and 1 elsewhere.

$$= -\frac{N}{1-p} \sum_{i=1}^{N} \int_{-\infty}^{y_i} w_i(s) ds + \frac{N}{p} \sum_{i=1}^{N} \int_{y_i}^{\infty} w_i(s) ds$$

Setting the above expression equal to zero we obtain the mle.

$$\hat{p} = P(y^* = 1) = \frac{\sum_{i=1}^N \int_{y_i}^\infty w_i(s)ds}{\sum_{i=1}^N \int_{-\infty}^\infty w_i(s)ds} \tag{3.17}$$

Note that when $w_i(s) = \frac{1}{N}I(0 < s < 1)$ equation 3.17 is equivalent to equation 3.13.

Maximum likelihood estimation for $f(s|y^*)$ takes more care. Since it is a density function, we will be obtaining a maximum function estimate. The component of the log-likelihood concerning $f(s|y^*)$ is

$$\ell(f(s|y^*)) = \sum_{i=1}^N \int_{-\infty}^\infty Nw_i(s)\left(1_{y_i^*(s)=0}\log f(s|0) + 1_{y_i^*(s)=1}\log f(s|1)\right)ds$$

.

I will obtain mles for $f(s|0)$ and $f(s|1)$ separately. For $\hat{f}(s|0)$ maximizing

$$\sum_{i=1}^N \int_{-\infty}^\infty Nw_i(s)1_{y_i^*(s)=0}\log f(s|0)ds \tag{3.18}$$

subject to the constraint

$$1 - \int_{-\infty}^\infty f(s|0)ds = 0 \tag{3.19}$$

results in an mle for $f(s|0)$. Utilizing the Lagrange multiplier λ, this constrained maximization is equivalent to maximizing

$$\begin{aligned}
&\sum_{i=1}^N \int_{-\infty}^\infty Nw_i(s)1_{y_i^*(s)=0}\log f(s|0)ds + \lambda\left(1 - \int_{-\infty}^\infty f(s|0)ds\right) \\
&\quad = \int_{-\infty}^\infty \left[\sum_{i=1}^N Nw_i(s)1_{y_i^*(s)=0}\right]\log f(s|0)ds + \lambda\left(1 - \int_{-\infty}^\infty f(s|0)ds\right) \\
&\quad = \int_{-\infty}^\infty \left[\left[\sum_{i=1}^N Nw_i(s)1_{y_i^*(s)=0}\right]\log f(s|0) - \lambda f(s|0)\right]ds + \lambda
\end{aligned} \tag{3.20}$$

Maximizing equation 3.20 is equivalent to maximizing the integrand pointwise with respect to $f(s|0)$ for each s. Differentiating the integrand and setting it equal to zero gives the mle.

$$\sum_{i=1}^N Nw_i(s)1_{y_i^*(s)=0}\frac{1}{\hat{f}(s|0)} - \lambda = 0$$

$$\Rightarrow \hat{f}(s|0) = \frac{\sum_{i=1}^{N} N w_i(s) 1_{y_i^*(s)=0}}{\lambda} \tag{3.21}$$

Combining the result in equation 3.21 with the constraint in 3.19 I can substitute the value of the Lagrange multiplier λ, the normalizing constant for $f(s|0)$.

$$\hat{f}(s|0) = \frac{\displaystyle\sum_{i=1}^{N} N w_i(s) 1_{y_i^*(s)=0}}{\displaystyle\int_{-\infty}^{\infty} \sum_{i=1}^{N} N w_i(s) 1_{y_i^*(s)=0} ds} = \frac{\displaystyle\sum_{i=1}^{N} w_i(s) 1_{y_i>s}}{\displaystyle\sum_{i=1}^{N} \int_{-\infty}^{y_i} w_i(s) ds} \tag{3.22}$$

Similar calculations for the mle of $f(s|1)$ yield

$$\hat{f}(s|1) = \frac{\displaystyle\sum_{i=1}^{N} w_i(s) 1_{y_i<s}}{\displaystyle\sum_{i=1}^{N} \int_{y_i}^{\infty} w_i(s) ds} \tag{3.23}$$

For discrete x maximum likelihood estimates are obtainable in a similar fashion to p. The portion of the likelihood for $f(x|0)$ takes on a d-dimensional multinomial-like form, the probability of taking on each possible state denoted as $P(x^{(j)}|0)$.

$$\begin{aligned} \ell(f(x|0)) &= \sum_{i=1}^{N} \int_{-\infty}^{\infty} N w_i(s) 1_{y_i^*(s)=0} \log \left[P(x^{(1)}|0)^{1_{x_i=x^{(1)}}} \cdots P(x^{(d)}|0)^{1_{x_i=x^{(d)}}} \right] ds \\ &= \sum_{i=1}^{N} \log \left[P(x^{(1)}|0)^{1_{x_i=x^{(1)}}} \cdots P(x^{(d)}|0)^{1_{x_i=x^{(d)}}} \right] \int_{-\infty}^{y_i} N w_i(s) ds \end{aligned} \tag{3.24}$$

But 3.24 is just a weighted multinomial log-likelihood with each observation weighted by $\int_{-\infty}^{y_i} N w_i(s) ds$. The mles then are simply

$$\hat{P}(X = x^{(j)}|Y^* = 0) = f(x|0) = \frac{\sum_{i:x_i=x^{(j)}} \int_{-\infty}^{y_i} w_i(s) ds}{\sum_{i=1}^{N} \int_{-\infty}^{y_i} w_i(s) ds}. \tag{3.25}$$

When X is a continuous predictor the estimation of $f(x|y^*)$ requires discretization or non-parametric density smoothing. For density smoothing of $f(x|0)$ we simply need to smooth the histogram of the observed x_i's with weights $\int_{-\infty}^{y_i} w_i(s) ds$. For $f(x|1)$ we use weights $\int_{y_i}^{\infty} w_i(s) ds$.

The estimators for the parameters of the naïve Bayes regression model also have another interpretation and an alternate derivation. Let us sample an observation from D such that the probability of selecting observation i is $P(i) \propto \int_{-\infty}^{\infty} w_i(s)ds$. Then let us sample an S from $P(s|i) \propto w_i(s)$. Using these two properties that define a sampling scheme, we can derive the same estimators using standard probability arguments. For example,

$$\begin{aligned}
\hat{P}(Y^* = 1) &= \sum_{i=1}^{N} P(Y_i^*(S) = 1|i)P(i) \\
&= \sum_{i=1}^{N} \left(\int_{-\infty}^{\infty} P(S > y_i|s, i)P(s|i)ds \right) P(i) \\
&= \sum_{i=1}^{N} \left(\int_{-\infty}^{\infty} I(s > y_i) \frac{w_i(s)}{\int_{-\infty}^{\infty} w_i(s')ds'} ds \right) \int_{-\infty}^{\infty} w_i(s)ds \\
&= \int_{y_i}^{\infty} w_i(s)ds.
\end{aligned}$$

3.6 The boosted naïve Bayes algorithm

The establishment of weight functions on each observation leads directly to the application of boosting. The manipulation of weights is a central idea of boosting and, as previously mentioned, this manipulation improves misclassification rates and, therefore in this application, regression error.

When we constrain $Y \in [0, 1]$ as in Freund and Schapire (1997), the weight functions for each observation on the first iteration may be uniform on [0,1]. Extending this method from [0,1] to the real line involves modifying the weight function so that they have finitely integrable tail (e.g. a function that decays exponentially from $s = y_i$) as in 3.14. We suggest initially using Laplace distribution weight functions of the form $w_i(s) \propto \exp -\frac{1}{\sigma}|s - y_i|$. Letting σ be large with respect to the spread of the data (so that the Laplace distribution is fairly flat) may let the boosting algorithm drive the modification of the weight functions. Freund and Schapire (1997) consider only $Y \in [0, 1]$ and propose initializing the weight function to be $w_i(s) = |s - y_i|$. This seems to be a poor choice of weight function since it ties the weight function to be 0 at $y = y_i$ and increases the weight on the region far from y_i. The most difficult region to classify must be in the neighborhood around y_i. If the classifier performs well at all then predicting whether y_i is smaller than s when s is much larger than y_i should be an easy task. The usual idea behind boosting is to downweight the easy to classify regions. Little to our surprise, in experiments with the algorithm when initialized to be uniform on [0,1], boosting increased the mass of the weight function in the neighborhood between the predicted y's and the true y_i's, the region of misclassification in D^*. The functions almost always moved to a Laplace-like form. This phenomenon is precisely opposite to Freund and Schapire's choice of initial weighting.

The total weighted empirical misclassification rate on iteration t is

$$\begin{aligned} \epsilon_t &= P_{D^*}(Y^*(S) \neq h_t^*(X,S)) \\ &= \sum_{i=1}^{N} \left| \int_{y_i}^{\hat{y}_i} w_i(s)ds \right| . \end{aligned} \tag{3.26}$$

Figure 3.1 compiles the preceding results into the boosted naïve Bayes regression algorithm. The reweighting step 4 of the algorithm can be rather complicated since the naïve Bayes classifier puts out a probabilistic prediction. If we abandon the added information available in the probability estimates in step 4 and instead merely use a threshold classification rule $I(\hat{P}(Y^* = 1|x_i, s) > \frac{1}{2})$ then the weight update is much simpler. With this 0-1 prediction the update step scales the weight function by β except on the interval between y_i and $\hat{y}_i$ (note that the discontinuity in the indicator function occurs at $\hat{y}_i$). This is the update scheme used in *AdaBoost.R*. To implement this alternate scheme, the algorithm stores $\hat{y}_i^{(t)}$ and β_t on each boosting iteration. The integrals in the estimation of the naïve Bayes model and the integral at step 2 of the boosting procedure then become computable in closed form as integrals of piecewise scaled sections of the Laplace distribution.

```
Input: sequence of examples $\{(x_1, y_1), \cdots, (x_n, y_n)\}$ where $y_i \in \Re$ and $T$ is the number of boosting iterations
Initialize: $w_i(s) \propto \exp(\frac{1}{\sigma}|s - y_i|)$, a Laplace density with mean $y_i$ and scale $\sigma$
For $t = 1, 2, \cdots, T$
    1. Using $w_i(s)$, estimate the components of the naïve Bayes regression model, $h_t(x)$.
    2. Calculate the loss of the model as $\epsilon_t = \sum_{i=1}^{N} \left| \int_{y_i}^{h_t(x_i)} w_i(s)ds \right|$.
    3. Set $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
    4. Update the weight functions according to the AdaBoost scheme as
       $w_i^{(t+1)}(s) = \begin{cases} w_i^t(s)\beta_t^{1-P(Y^*=1|X,s)} & s \le y_i \\ w_i^t(s)\beta_t^{P(Y^*=1|X,s)} & s > y_i \end{cases}$   (3.27)
    5. Normalize the weights so that $\sum_{i=1}^{N} \int_{-\infty}^{\infty} w_i(s)ds = 1$.
Output the model:
$\hat{Y} = \inf_y \left\{ y : \sum_{t=1}^{T} \alpha_t \log \frac{P_S^t(y|Y^*=0)}{P_S^t(y|Y^*=1)} \le \sum_{t=1}^{T} \alpha_t \log \frac{P^t(Y^*=1)}{P^t(Y^*=0)} + \sum_{j=1}^{d} \sum_{t=1}^{T} \alpha_t \log \frac{P^t(X_j|Y^*=1)}{P^t(X_j|Y^*=0)} \right\}$
where $\alpha_t = \frac{\log \beta_t}{\sum_{t=1}^{T} \log \beta_t}$.
```

Figure 3.1: The boosted naïve Bayes regression algorithm

3.7 Experimental results

The experimental work with the boosted naïve Bayes regression uses a discrete approximation to the algorithm developed in the previous section. We actually constructed D^* with S as a finite sequence of evenly spaced values ($m = 100$ in our experiments), initialized the weight functions to be uniform on [0,1], and fit the boosted naïve Bayes classifier. Experimenting in this way gave us some intuition on the performance of the method and how *AdaBoost* modifies the weights of the hard to classify regions. We show empirically that the boosted naïve Bayes regression can capture many interesting regression surfaces. Because our experimentation used this discrete approximation, we are only able to handle a response bounded on [0,1]. Therefore, in all experiments we shifted and scaled the response to the unit interval. For the continuous predictors we used a non-parametric density estimator (LOCFIT, Loader 1997) to estimate $P(X_j|Y^*)$ and $P(S|Y^*)$. LOCFIT is a local density estimator that can handle observation weights.

For each simulated test function we generated 100 observations as a training dataset and 100 observations for a validation set. For the two real datasets we randomly selected half of the observations as training and the remaining half as a validation set. From the training dataset we fit the boosted naïve Bayes regression model, a least squares plane, a generalized additive model, and a CART model. We replicated each experiment ten times and measured the performance on the validation set using mean squared bias, $\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$.

As the boosting iterations proceeded $\log \beta_t$ always approached zero so that each additional iteration contributed less and less to error improvement. We continued to boost until $\log \beta_t$ was fairly small. This stopping criteria generally did not affect the performance on the validation set.

We tested using the following functions.

1. A plane

$$y = 0.6x_1 + 0.3x_2 + \epsilon, \text{ where } \epsilon \sim N(0, 0.05)$$
$$x_j \sim U[0,1], j = 1, 2$$

2. Friedman #1 (Friedman, Grosse, and Stuetzle 1983)

$$y = 10\sin(\pi x_1 x_2) + 20(x_3 - \frac{1}{2})^2 + 10x_4 + 5x_5 + N(0,1)$$
$$x_j \sim U[0,1], j = 1, \ldots, 10$$

3. Friedman #2 (Friedman 1991)

$$y = \left(x_1^2 + \left(x_2 x_3 - \frac{1}{x_2 x_4}\right)^2\right)^{\frac{1}{2}} + N(0, \sigma)$$

4. Friedman #3 (Friedman 1991)

$$y = \tan^{-1}\frac{x_2 x_3 - \frac{1}{x_2 x_4}}{x_1} + N(0, \sigma)$$

For both 3 and 4 σ is such that the true underlying function explains 91% of the variability in y and

$$\begin{aligned} x_1 &\sim U[0, 100] \\ x_2 &\sim U[40\pi, 560\pi] \\ x_3 &\sim U[0, 1] \\ x_4 &\sim U[1, 11] \end{aligned}$$

5. Bank dataset (George and McCulloch 1993)

   This dataset contains financial information on 233 banks in the greater New York area. We selected eleven of the variables for predicting the number of new accounts sold in a fixed time period.

6. Body fat dataset (Penrose, Nelson, and Fisher 1985)

   This dataset contains physical measurements on 252 men. From a set of non-invasive body measurements we attempt to predict body fat percentage.

For all the datasets we linearly scaled the response so that y fell in the interval [0,1].

3.8 Performance results

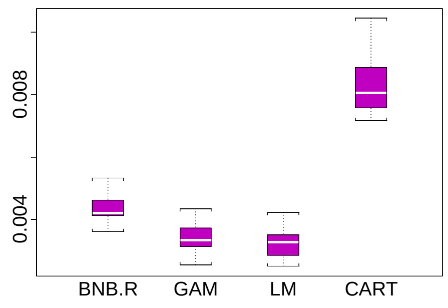

Figure 3.2: Hyperplane

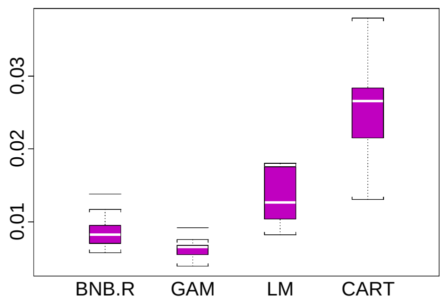

Figure 3.3: Friedman #1

In the first example, the plane, the least-squares linear model is the best predictive model to fit. Naturally, any other model cannot outperform the ordinary least squares plane in terms of generalization error, but we do desire that the boosted naïve Bayes regression model would still perform relatively well. Figure 3.2 shows that indeed BNB.R did not perform as well as LM or GAM but its performance was satisfactory.

Friedman, Grosse, and Stuetzle (1983) proposed Friedman #1 to test learning noisy functions that are additive with interactions. Furthermore, it introduces five

Table 3.3: Performance results - simulated datasets

Function	Model	Mean squared bias	standard deviation
Plane	BNB.R	0.0044	0.0005
	GAM	0.0034	0.0005
	LM	0.0033	0.0005
	CART	0.0083	0.0010
Friedman #1	BNB.R	0.0087	0.003
	GAM	0.0064	0.001
	LM	0.0132	0.003
	CART	0.0248	0.007
Friedman #2	BNB.R	0.0072	0.002
	GAM	0.0060	0.002
	LM	0.0132	0.003
	CART	0.0091	0.002
Friedman #3 $N = 100$	BNB.R	0.0106	0.002
	GAM	0.0099	0.001
	LM	0.0182	0.003
	CART	0.0194	0.006
Friedman #3 $N = 200$	BNB.R	0.009	0.0016
	GAM	0.009	0.0011
	LM	0.016	0.0025
	CART	0.012	0.0018

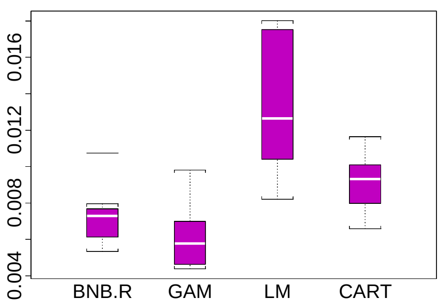

Figure 3.4: Friedman #2

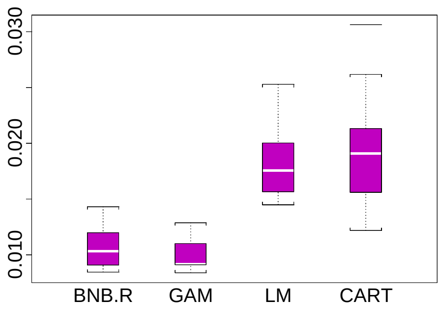

Figure 3.5: Friedman #3

variables, x_6 to x_{10}, which are purely extraneous variables. We found that BNB.R outperformed the linear model and CART but GAM still outperformed BNB.R. Figure 3.3 shows the performance over 10 validation datasets.

Friedman (1991) proposed learning functions Friedman #2 and Friedman #3, the impedance and phase shift of a specific circuit where x_1, x_2, x_3, x_4 relate to a resistor, a generator, an inductor, and a capacitor. Figure 3.4 and figure 3.5 show the performance on Friedman #2 and #3. On function Friedman #2 BNB.R outperformed the linear model and CART and performed a little worse than GAM. Among all the simulated function experiments BNB.R was most competitive with GAM on Friedman #3.

Table 3.3 summarizes the performance on the simulated datasets and table 3.4 the real datasets. On the bank dataset GAM performed especially poorly and BNB.R was third to LM and CART. Unlike the simulated examples, these datasets do not have a controlled error structure. At this point we are uncertain how sensitive BNB.R is to very noisy data. We also briefly investigated how changes in the sample size affect the performance by including a second analysis of Friedman #3 with $N = 200$. CART, as a Bayes risk consistent regression procedure, naturally gains substantially in performance. GAM and BNB.R improve slightly, although not by a significant

Table 3.4: Performance results - real datasets

Function	Model	Mean squared bias	standard deviation
Bank	BNB.R	0.010	0.0031
	GAM	0.018	0.0201
	LM	0.005	0.0017
	CART	0.009	0.0008
Body fat	BNB.R	0.017	0.002
	GAM	0.014	0.005
	LM	0.010	0.001
	CART	0.014	0.002

amount.

3.9 Conclusions

In this chapter we brought together ideas from boosting, naïve Bayes learning, additive modeling, and induced regression models from classification models. We derived the naïve BNB.R algorithm to fit a boosted naïve Bayes regression model. Using a discrete approximation to BNB.R, we compared its performance to three other interpretable multivariate regression procedures that are widely used. Although the results show that at this stage of development BNB.R is not as competitive as other, more established methods, we believe the novelty and the unexpected satisfactory performance warrants further research.

The most important aspect from a research perspective is that applying the boosted naïve Bayes classifier in this fashion provides an early link between the advances boosting has made for classification problems to its potential application in regression contexts. Changes in the base classifier, an improved implementation, and further research into the properties of boosting may introduce a new, rich class of regression models.

Chapter 4

BOOSTING EXPONENTIAL FAMILY AND PROPORTIONAL HAZARDS REGRESSION MODELS

The chapter discusses the extension of boosting methods to exponential family and proportional hazards regression models commonly fit using some form of generalized linear models (GLM) (McCullagh and Nelder 1989) or generalized additive models (GAM) (Hastie and Tibshirani 1990). Using gradient ascent optimization methods we can estimate regression functions for many models used in standard statistical practice. Specifically this chapter will consider boosted versions of the generalized linear model, such as logistic, Poisson, Gaussian, as well as proportional hazards regression including the Cox model. The modular structure of boosting algorithms also exposes these models to robust regression techniques, makes implementations for massive datasets feasible, and can estimate non-parametric regression functions utilizing existing fitting algorithms for their linear counterparts.

4.1 Exponential family models

Oftentimes in data analyses we wish to model or estimate the expected value of a random variable, Y, as a function of some observable covariates or predictors, x. To aid in the estimation of $\mathrm{E}(Y|x)$, we assume a distributional form for $Y|x$ and a class of functions of which $\mathrm{E}(Y|x)$ is assumed to be a member. Assuming that the distribution of $Y|x$ is a member of the exponential family and that $\mathrm{E}(Y|x)$ is an invertible function of a linear combination of the covariates, we arrive at the generalized linear model. Nelder and Wedderburn (1972) originally used the phrase "generalized linear model" and McCullagh and Nelder (1989) set up the framework for the generalized linear model as follows.

1. *Distributional assumption*: $Y|x$ has a distribution in the exponential family of the form

$$f(y|\theta,\phi,x) = \exp\left(\frac{y\theta - b(\theta)}{a(\phi)} - c(y,\phi)\right) \tag{4.1}$$

   where $b(\theta)$ is the cumulant generating function so that $b'(\theta) = \mathrm{E}(Y|x) = \mu$.

2. *Model for the mean*: The conditional mean of the response is the modeled functional.

$$\mu_i = \mathrm{E}(Y|X_i)$$

3. *Systematic component*: The covariates, $x_1, x_2, \cdots, x_d$, form a linear predictor

$$\eta_i = \sum_{j=1}^{d} \beta_j x_{ij} \tag{4.2}$$

4. *A link function*: The link function is a monotone, differentiable function relating the linear predictor to the response.

$$g(\mu_i) = \eta_i$$

   When g is such that $\theta = \eta$, it is known as the canonical link function.

Many useful statistical models fall into this class including linear regression, linear logistic regression, log-linear models, and, in some form, linear proportional hazards regression. Fitting a model under this framework is equivalent to selecting the β_i's, usually by maximizing the appropriate likelihood. The linearity assumption is convenient since it compresses all the information concerning a particular variable into one number, lending easy interpretability to the model. Furthermore, estimation in linear models is extremely efficient.

When accurate prediction is our goal the linear assumption can be a potential weakness of this method. If the true, underlying regression function is non-linear then GLM introduces a bias that more data cannot correct. An ideal regression procedure would allow $\mathrm{E}(Y|x)$ to take on an arbitrary unknown functional form, $F(x)$, and then attempt to estimate F non-parametrically. Clearly in practice such flexibility is infeasible since we would need to estimate an infinite dimensional parameter, F, using a finite dataset. In this case, a unique function estimate, $\hat{F}$, that maximizes the likelihood does not exist.

Generalized additive models (Hastie and Tibshirani 1990) extend the generalized linear model framework by loosening the linearity assumption in equation 4.2. Rather than assume that $g(\mu)$ is a linear combination of the predictors, GAM assumes that

$$g(\mu_i) = \sum_{j=1}^{d} f_j(x_{ij}) \tag{4.3}$$

where the d functions $f_j()$ are unknown functions of only one of the predictor variables. To ensure a unique solution for the maximum likelihood estimates of the f_j's GAM forces them to be smooth and to have 0 average. In many practical applications, smoothness is reasonable to assume and only the degree of smoothness need be considered (usually set by cross-validation). Algorithms for estimating the $f_j()$'s involve the use of smoothers and the backfitting algorithm (Friedman and Stuetzle 1981).

Although GAM relaxes the linearity assumption it still offers interpretability. However, it does contain an uncorrectable bias when the true underlying function is not additive. Tree structured regressors such as CART (Breiman, Friedman, Olshen, and Stone 1984) may be used to create risk-consistent non-parametric estimators of F. The greedy recursive partitioning strategy chooses, among all possible splits along covariate axes, the split that offers the greatest increase in the likelihood. The depth of the tree controls the degree of fit to the training data. As data accumulates and the depth of the tree increases, the recursive partitioning estimator converges to the true underlying regression function (Gordon and Olshen 1984). Tree structured regressors have been applied to survival problems in Segal (1988) and Ripley and Ripley (1999). However, tree regressors suffer from "high variability" (Breiman 1996b). That is, small changes in the dataset may produce very different trees and predictions. This can be mitigated by variance reduction techniques such as bagging (Breiman 1996a, Friedman and Hall 1999) and Bayesian methods (Chipman, George, and McCulloch 1998, Denison, Mallick, and Smith 1996). Although tree methods overcome the additive assumption necessary for GAM, it produces discontinuous regression function estimates (bagged trees can be smoother). Multivariate adaptive regression splines or MARS (Friedman 1991) uses adaptive knot placement rather than discontinuity placement to produce function estimates. MARS is adaptable to generalized linear models and survival models (Intrator and Kooperberg 1995).

4.2 Boosting methods for regression problems

Boosting, rapidly made popular in the machine learning community, is a topic slowly making its way into mainstream statistics research. Computational learning theorists desired a method for transforming a collection of weak classifiers into one strong classifier (Schapire 1990, Freund 1995). This work culminated in the work by Freund and Schapire (1997) who introduced the *AdaBoost* algorithm. They discovered an algorithm which sequentially fits "weak" classifiers to different weightings of the observations in a dataset. Those observations that the previous classifier poorly predicts receive higher weight on the next iteration. The final *AdaBoost* classifier is a weighted average of all the weak classifiers. In a wide variety of classification problems, their weighting scheme and final classifier merge have proven to be an effective method for reducing bias and variance, and improving misclassification rates (Bauer and Kohavi

1999, Dietterich 1998). Empirical evidence has shown that the base classifier can be fairly simplistic (shallow classification trees) and yet, when boosted, can capture complex decision boundaries (Breiman 1998).

At the conclusion of their paper, Freund and Schapire (1997) outline their ideas for applying the *AdaBoost* algorithm to regression problems. However, Drucker (1997) first proposed and tested practical methods for boosting regression. His *ad hoc* method followed the same spirit of the *AdaBoost* algorithm by repeatedly performing weighted tree regression followed by upweighting the poorly predicted observations and downweighting the well predicted ones. Compared with fitting just one tree, his method seems to be competitive. Breiman (1997) suggests how one might deal with boosting regression problems with his *arc-gv* methodology. His more recent work on adaptive bagging (Breiman 1999), which is quite similar to boosting ideas, appears to be a very promising direction. Both boosting and adaptive bagging proceed similarly and both show improvements in terms of bias and variance.

Ridgeway, Madigan, and Richardson (1999) proposed mapping the regression problem into a classification problem, applying their boosted naïve Bayes classifier, and transforming the resulting classifier back as a regression function. They showed that this method fits an additive model similar to the form 4.3 but also estimates a transformation of the response variable. Their empirical results show that it performs similarly to GAM and outperforms CART in four simulated examples but performs slightly worse than CART on one of the two real datasets investigated.

From the statistician's point of view, a breakthrough in boosting occured in Friedman, Hastie, and Tibshirani (1998). Although still focused on classification problems, they showed that Freund and Schapire's *AdaBoost* algorithm is an optimization method for finding a classifier that minimizes a particular exponential loss function. Other authors also have made the connection between boosting and optimization (Breiman 1997, Mason, Baxter, Bartlett, and Frean 1999). Friedman et al. proceeded to show that this exponential loss function used in *AdaBoost* is closely related to the Bernoulli likelihood and developed a new boosting algorithm that finds a classifier that maximizes a Bernoulli likelihood. The important finding of this work is that it links work in computational learning theory to a likelihood based method of standard statistical practice.

At the heart of their work, Friedman et al. wish to construct a classifier that takes in covariates, x, and attempts to predict a binary outcome, $y \in \{0, 1\}$. The classifier, $\hat{F}(x)$, returns a negative value to predict a 0 and returns a positive value to predict a 1. Ideally we want this $\hat{F}(x)$ to have large Bernoulli likelihood on average over all future observables. If we let $J(F)$ be the expected Bernoulli likelihood of a particular classifier, F, then mathematically $\hat{F}(x)$ should be

$$\begin{aligned}\hat{F}(x) &= \arg\max_{F(x)} J(F) \\ &= \arg\max_{F(x)} \mathrm{E}_{y|x}\left[yF(x) - \log\left(1 + e^{F(x)}\right) \middle| x\right] \quad (4.4)\end{aligned}$$

If we have a current estimate for $F(x)$ and we wish to improve upon it, we might add a new function to F slightly modifying it to further increase $J(F)$, the likelihood shown in 4.4. We can select one of many optimization methods to find an f that increases $J(F + f)$. Friedman et al. chose to derive a Newton algorithm. That is, given a current estimate for $F(x)$ the Newton update for $\hat{F}(x)$ is

$$\begin{aligned}\hat{F}(x) &\leftarrow \hat{F}(x) - \frac{\frac{\partial}{\partial f} J(\hat{F} + f)|_{f=0}}{\frac{\partial^2}{\partial f^2} J(\hat{F} + f)|_{f=0}} \quad (4.5) \\ &= \hat{F}(x) + \mathrm{E}_w\left[\frac{y - p(x)}{p(x)(1 - p(x))}|x\right] \quad (4.6) \\ &\quad \text{where } w(x, y) = p(x)(1 - p(x)) \text{ and } p(x) = \frac{1}{1 + e^{-\hat{F}(x)}} \quad (4.7)\end{aligned}$$

Newton optimization, therefore, tells us to repeatedly substitute weighted conditional class probability estimates to improve $F(x)$. Note that the weighting in 4.7 implies that largest weights are piled onto the difficult to classify observations, those near $p(x) = \frac{1}{2}$. This is similar in spirit to the *AdaBoost* algorithm. Since we do not know the value of the expectations in 4.6 we can approximate them by choosing one of our favorite regression methods. Substituting a linear model in 4.6 for the weighted expectation reduces to the iteratively reweighted least squares algorithm (Green 1987) for fitting linear logistic regression models. Friedman et al. use CART regression in their experimental results and show substantial improvement in accuracy over a non-boosted CART classifier.

Once boosting and likelihood methods become related the door opens to many other statistical models. And so the findings of Friedman, Hastie, and Tibshirani (1998) lead us directly into methodology for generating boosting algorithms for a large class of other likelihood based models.

4.2.1 Friedman's gradient boosting machine

Friedman (1999a) and the companion paper Friedman (1999b) extended the work of Friedman, Hastie, and Tibshirani (1998) and laid the ground work for a new generation of boosting algorithms. Using the connection between boosting and optimization made in Friedman et al. (1998), this new work proposes the Gradient Boosting Machine.

In any function estimation problem we wish to find a regression function, $\hat{F}(x)$, that minimizes the expectation of some loss function, $\Psi(y, F)$, as shown in equation 4.11.

$$\begin{aligned}\hat{F}(x) &= \arg\min_{F(x)} \mathrm{E}_{y,x}\Psi(y, F(x)) \\ &= \arg\min_{F(x)} \mathrm{E}_x \left[\mathrm{E}_{y|x}\Psi(y, F(x))\Big| x\right] \qquad (4.11)\end{aligned}$$

We will focus on finding estimates of $F(x)$ such that

$$\hat{F}(x) = \arg\min_{F(x)} \mathrm{E}_{y|x}\left[\Psi(y, F(x))|x\right] \qquad (4.12)$$

Parametric regression models assume that $F(x)$ is a function with a finite number of parameters, β, and estimates them by selecting those values that minimize a loss

Initialize $\hat{F}(x)$ to be a constant, $\hat{F}(x) = \arg\min_\rho \sum_{i=1}^N \Psi(y_i, \rho)$.

For t in $1, \cdots, T$ do

1. Compute the negative gradient as the working response

$$z_i = -\frac{\partial}{\partial F(x_i)} \Psi(y_i, F(x_i)) \bigg|_{F(x_i)=\hat{F}(x_i)} \tag{4.8}$$

2. Fit a regression model, $f(x)$, predicting z_i from the covariates x_i.

3. Choose a gradient descent step size as

$$\rho = \arg\min_\rho \sum_{i=1}^N \Psi(y_i, \hat{F}(x_i) + \rho f(x_i)) \tag{4.9}$$

4. Update the estimate of $F(x)$ as

$$\hat{F}(x) \leftarrow \hat{F}(x) + \rho f(x) \tag{4.10}$$

Figure 4.1: Friedman's Gradient Boost algorithm

function (i.e. squared error loss) over a training sample of N observations on (y, x) pairs as in 4.13.

$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{N} \Psi(y_i, F(x_i; \beta)) \tag{4.13}$$

When we wish to estimate $F(x)$ non-parametrically the task becomes more difficult. Again we can proceed similarly to 4.5 and modify our current estimate of $F(x)$ by adding a new function $f(x)$ in a greedy fashion. Letting $F_i = F(x_i)$, we see that we want to decrease the N dimensional function

$$\begin{aligned} J(\mathbf{F}) &= \sum_{i=1}^{N} \Psi(y_i, F(x_i)) \\ &= \sum_{i=1}^{N} \Psi(y_i, F_i). \end{aligned} \tag{4.14}$$

The negative gradient of $J(\mathbf{F})$ indicates the direction of the locally greatest decrease in $J(\mathbf{F})$. Gradient descent would then have us modify $\mathbf{F}$ as

$$\hat{\mathbf{F}} \leftarrow \hat{\mathbf{F}} - \rho \nabla J(\mathbf{F}) \tag{4.15}$$

where ρ is the size of the step along the direction of greatest descent. Clearly, this step alone is far from our desired goal. First, it only fits F at values of x for which we have observations. Second, it does not take into account that observations with similar x are likely to have similar values of $F(x)$. Both these problems would have disastrous effects on generalization error. However, Friedman suggests selecting a class of functions that use the covariate information to approximate the gradient. This line of reasoning produces his Gradient Boosting algorithm shown in figure 4.1. At each iteration the algorithm determines the direction, the gradient, in which it needs to improve the fit to the data and selects a particular model from the allowable class of functions that is in most agreement with the direction. In the case of squared-error loss, $\Psi(y_i, F(x_i)) = \sum_{i=1}^{N}(y_i - F(x_i))^2$, this algorithm corresponds exactly to residual fitting. Friedman also uses this algorithm to develop new boosting algorithms for robust regression with least absolute deviation and Huber loss functions (Huber 1964).

As in the following sections I will apply these methods directly to likelihood based models. We will transform 4.12 so that it involves a maximization instead of a minimization and consider various log-likelihood functions for $\Psi(y, F)$.

4.2.2 Generalized boosted models by Fisher scoring

Now having the machinery to extend boosting to likelihood based models, we can return to the problem of boosting models in the exponential family of section 4.1. In this section we will derive a boosting algorithm based on a variant of the Newton-Raphson optimizer known as the scoring method (Fisher 1935, Finney 1971).

The extension that generalized boosted models makes to GLM is to further loosen the linear assumption in equation 4.2, only assuming that

$$g(\mu_i) = F(x_{i1}, x_{i2}, \ldots, x_{id}). \tag{4.16}$$

The boosting algorithm searches in a greedy fashion for a function $\hat{F}$ that maximizes the likelihood under the assumed model.

Depending on how we restrict the boosting procedure, modeling $g(\mu)$ in this fashion can also remove the burden of choosing a link function. The link function is only a vehicle to transform values from the linear or additive model scale to the scale of the conditional expectation. Under 4.16 it does not matter whether we choose to model via a link function, $\mu_i = g^{-1}(F(x_i))$, or directly model the mean, $\mu_i = F(x_i)$. In practice, however, it may be reasonable to restrict F to be additive with low order interactions with an interpretable link function such as the log-odds for the logistic model.

To derive a boosting algorithm for the general case of exponential family models we simply need to select a model by choosing a specific cumulant generating function, $b(\theta)$, and a link function, $g(\mu)$, relating the regression function, $F(x)$, to the conditional mean, μ. The following proposition derives the necessary results to produce a boosting algorithm for the exponential family models and figure 4.2 summarizes this result.

Proposition 4.1 *The generalized boosted model algorithm is a Fisher scoring algorithm that optimizes the sample expected log-likelihood of an exponential family regression model.*

Proof: Let $b(\theta)$ be the cumulant generating function, μ be the expected value of Y, and $g(\mu)$ the link function between the conditional mean and $F(x)$. The expected log-likelihood for the exponential family model is

$$\mathrm{E}_{Y|x}\ell(\theta,\phi|y,x) = \mathrm{E}_{Y|x}\left[\frac{1}{a(\phi)}(y\theta - b(\theta))\Big|x\right]. \tag{4.21}$$

Using the fact that $\mathrm{E}\frac{\partial\ell}{\partial\theta} = 0$ and $\mathrm{E}\frac{\partial^2\ell}{\partial\theta^2} + \left(\mathrm{E}\frac{\partial\ell}{\partial\theta}\right)^2 = 0$, we can show that $b'(\theta) = \mu$ and $\mathrm{Var}(Y) = b''(\theta)a(\phi)$. The term $b''(\theta)$ is the variance function which we will write as a function of μ, $V(\mu)$.

We will initialize the estimated regression function, $F(x)$, to be a constant across all x. We can start with the constant that maximizes 4.21. The value of θ that maximizes 4.21 satisfies the equation $b'(\theta) = \bar{y}$. Since $\mu = b'(\theta)$ and $g(\mu) = F(x)$ then we should initialize $\hat{F}(x) = g(\bar{y})$ for all values of x.

Given the current estimate of the regressor $\hat{F}(x)$ we would like to improve upon it by adding a new function $f(x)$ so that the expected likelihood in 4.21 increases. That is, initially assuming that $\eta = g(\mu) = \hat{F}(x)$ we will seek an $f(x)$ so that when $g(\mu) = \hat{F}(x) + f(x)$ the likelihood increases. The Newton-Raphson variant known as the scoring method suggests modifying the current regressor as

$$\hat{F}(x) \leftarrow \hat{F}(x) - \frac{\mathrm{E}_{Y|x}\frac{\partial\ell(\hat{F}+f)}{\partial f}}{\mathrm{E}_{Y|x}\frac{\partial^2\ell(\hat{F}+f)}{\partial f^2}}. \tag{4.22}$$

The strategy that I will use here is to carry through with the denominator expectation to simplify computation, but preserve the expectation in the numerator for estimation

```
Initialize $\hat{F}(x) = g(\bar{y})$ for all values of $x$.
For $t$ in $1, \dots, T$ do
    1. Estimate the predictions as
       $\mu_i = g^{-1}(\hat{F}(x_i)).$   (4.17)
    2. Compute the working response
       $z_i = (y_i - \mu_i)g'(\mu_i).$   (4.18)
    3. Fit a regression model, $f(x)$, predicting $z_i$ using $x_i$ with weights
       $w_i = \frac{1}{g'(\mu_i)^2 V(\mu_i)}$   (4.19)
    4. With $\lambda \in (0, 1]$ as a learning rate parameter, update the boosted regressor as
       $\hat{F}(x) \leftarrow \hat{F}(x) + \lambda f(x)$   (4.20)
Return the final boosted regressor $\hat{F}(x)$.
```

Figure 4.2: The generalized boosted algorithm by Fisher scoring

with an empirically derived regression function. Starting with the numerator

$$\begin{aligned}
\mathrm{E}_{Y|x}\frac{\partial \ell}{\partial f} &= \mathrm{E}_{Y|x}\frac{\partial \ell}{\partial \theta}\frac{\partial \theta}{\partial \mu}\frac{\partial \mu}{\partial \eta}\frac{\partial \eta}{\partial f} \\
&= \mathrm{E}_{Y|x}\left[\frac{y-b'(\theta)}{a(\phi)}\frac{1}{b''(\theta)}\frac{1}{g'(\mu)}\cdot 1\Big|x\right] \\
&= \mathrm{E}_{Y|x}\left[\frac{y-\mu}{a(\phi)}\frac{1}{V(\mu)}\frac{1}{g'(\mu)}\Big|x\right]. \qquad (4.23)
\end{aligned}$$

In the above equation we use that fact that since $b'(\theta)=\mu$ then $\frac{\partial\theta}{\partial\mu}=\frac{1}{b''(\theta)}$.

Differentiating 4.23 with respect to f we obtain the denominator term.

$$\begin{aligned}
\mathrm{E}_{Y|x}\frac{\partial^2 \ell}{\partial f^2} &= \mathrm{E}_{Y|x}\left[\frac{\partial}{\partial \mu}\left(\frac{\partial \mathrm{E}_{Y|x}\ell}{\partial f}\right)\frac{\partial \mu}{\partial \eta}\frac{\partial \eta}{\partial f}\Big|x\right] \\
&= \mathrm{E}_{Y|x}\Bigg[\Bigg(-\frac{1}{a(\phi)}\frac{1}{V(\mu)}\frac{1}{g'(\mu)}- \\
&\qquad \left(\frac{y-\mu}{a(\phi)}\right)\left(\frac{g''(\mu)}{V(\mu)g'(\mu)^2}+\frac{V'(\mu)}{V(\mu)^2g'(\mu)}\right)\Bigg)\frac{1}{g'(\mu)}\cdot 1\Big|x\Bigg] \qquad (4.24)
\end{aligned}$$

Since $\mathrm{E}(Y)=\mu$, taking the expectation in this term causes the second (and most complex) part of the expression to vanish reducing 4.24 to

$$\mathrm{E}_{Y|x}\left[\frac{\partial^2\ell}{\partial f^2}\Big|x\right] = -\mathrm{E}_{Y|x}\left[\frac{1}{a(\phi)V(\mu)g'(\mu)^2}\Big|x\right] \qquad (4.25)$$

If we had assumed a canonical link function then the following holds

$$\begin{aligned}
g^{-1}(\theta) &= b'(\theta) \\
\Rightarrow g'(\mu) &= \frac{1}{b''(\theta)}=\frac{1}{V(\mu)} \qquad (4.26)
\end{aligned}$$

and so the same term vanishes without requiring the expectation. Under canonical link functions the Fisher scoring and Newton-Raphson algorithms are equivalent.

After simplifying the denominator term, we will preserve the expectation operator on this constant for the moment even though it involves no random variables. Substituting these results into the Fisher scoring algorithm in equation 4.22 we obtain

$$\hat{F}(x)\leftarrow\hat{F}(x)+\frac{\mathrm{E}\left[\frac{y-\mu}{a(\phi)V(\mu)g'(\mu)}\Big|x\right]}{\mathrm{E}\left[\frac{1}{a(\phi)V(\mu)g'(\mu)^2}\Big|x\right]}. \qquad (4.27)$$

Using the definition of weighted expectation, $\mathrm{E}_w f(y,x) = \frac{\mathrm{E}[w(x,y)f(x,y)]}{\mathrm{E}[w(x,y)]}$, we note that 4.27 is a weighted conditional expectation. We can rewrite 4.27 as

$$\begin{aligned}
\hat{F}(x) &\leftarrow \hat{F}(x) + \frac{\mathrm{E}\left[\frac{a(\phi)V(\mu)g'(\mu)^2}{a(\phi)V(\mu)g'(\mu)^2}\frac{y-\mu}{a(\phi)V(\mu)g'(\mu)}|x\right]}{\mathrm{E}\left[\frac{1}{a(\phi)V(\mu)g'(\mu)^2}|x\right]} \\
&= \hat{F}(x) + \frac{\mathrm{E}\left[\frac{1}{a(\phi)V(\mu)g'(\mu)^2}g'(\mu)(y-\mu)|x\right]}{\mathrm{E}\left[\frac{1}{a(\phi)V(\mu)g'(\mu)^2}|x\right]} \\
&= \hat{F}(x) + \mathrm{E}_w\left[g'(\mu)(y-\mu)|x\right] \text{ where } w = \frac{1}{a(\phi)V(\mu)g'(\mu)^2} \qquad (4.28)
\end{aligned}$$

When we have a finite sample we can estimate the weighted conditional expectation by any regression procedure (trees, smoothers, etc.) to update our estimated regression function. Note that since $a(\phi)$ is constant across all values of μ it vanishes when the weights are normalized. This yields the generalized boosted algorithm in figure 4.2.

$\square$

We can vary several of the steps in the GBM algorithm to improve its performance. Specifically noted here is the introduction of a learning rate parameter, λ, in 4.20. Section 4.4 will further discuss the learning rate as well as other modifications such as bagging for variance reduction and robust methods.

The link function is *canonical* if $g(\mu)$ is such that $\theta = \eta$. This occurs when $g^{-1}(\theta) = b'(\theta)$. Table 4.1 lists some common exponential models, their canonical link functions, and the associated variance function. Note that in these canonical cases $V(\mu)g'(\mu) = 1$ further simplifying the boosting algorithm. One specific instance of this algorithm is already known as shown in the following corollary to proposition 4.1.

Corollary 4.1 *The LogitBoost algorithm of Friedman, Hastie, and Tibshirani (1998) is a generalized boosted model algorithm for a canonical Bernoulli model.*

Table 4.1: Some canonical exponential family models

Distribution	Canonical link $g(\mu)$	Variance function $V(\mu)$
Bernoulli	$\log \frac{\mu}{1-\mu}$	$\mu(1-\mu)$
Normal	μ	1
Poisson	$\log \mu$	μ

Proof: To boost the canonical Bernoulli model we select $g(\mu) = \log \frac{\mu}{1-\mu}$ and $V(\mu) = \mu(1-\mu)$. Therefore, $g'(\mu) = \frac{1}{\mu(1-\mu)}$. The GBM algorithm then indicates that one should iterative perform weighted regression predicting $z_i = \frac{y_i - \mu}{\mu(1-\mu)}$ with weights $w_i = \mu(1-\mu)$. This is equivalent to the expression in 4.6, the *LogitBoost* algorithm.

$\square$

4.2.3 Generalized boosted models by gradient ascent

Section 4.2.2 extended the boosting methodology of Friedman, Hastie, and Tibshirani (1998) from logistic regression to an arbitrary exponential family model using Fisher scoring. This section will explore the use of gradient ascent methods akin to Friedman (1999a) for application to the exponential family model.

Again we will be searching for a regression function $\hat{F}(x)$ that maximizes

$$\mathrm{E}_{Y|x}\ell(\theta,\phi|y,x) = \mathrm{E}_{Y|x}\left[\frac{1}{a(\phi)}(y\theta - b(\theta))\Big|x\right]. \tag{4.29}$$

Given a finite training dataset we can approximate 4.29 by

$$\begin{aligned}\hat{\mathrm{E}}_{Y|x}\ell(\theta,\phi|y,x) &\approx J(F) \\ &= \sum_{i=1}^{N} \frac{1}{a(\phi)}(y\theta_i - b(\theta_i))\end{aligned} \tag{4.30}$$

where again $b'(\theta) = \mu$ and $g(\mu) = F(x)$. For the moment we will assume that each of the N observations has their own value for F, or $F(x_i) = F_i$. Later we will incorporate the covariates. When the set of F_i's take on specific values, the gradient evaluated at that point indicate the direction of greatest ascent. Gradient ascent methods suggest modifying the current values of the F_i's by

$$\mathbf{F} \leftarrow \mathbf{F} + \rho \nabla \mathbf{F} \tag{4.31}$$

where ρ is the step size in the gradient direction. Computing the gradient from 4.30 we obtain

$$\begin{aligned}
\frac{\partial J}{\partial F_i} &= \frac{\partial J}{\partial \theta_i}\frac{\partial \theta_i}{\partial \mu_i}\frac{\partial \mu_i}{\partial \eta_i}\frac{\partial \eta_i}{\partial F_i} \\
&= \frac{y_i - b'(\theta_i)}{a(\phi)}\frac{1}{b''(\theta_i)}\frac{1}{g'(\mu_i)} \cdot 1 \\
&= \frac{y_i - \mu_i}{a(\phi)V(\mu_i)g'(\mu_i)} \\
&\doteq z_i
\end{aligned} \tag{4.32}$$

where $\mu_i = g^{-1}(F(x_i))$.

Equation 4.31 says that we should add to the current value of each F_i a constant times the function in 4.32. Note that 4.32 is basically a scaled residual. At this stage we can incorporate the covariates by approximating this gradient using an arbitrary least squares regression procedure. That is, we can compress the gradient information and borrow strength from observations that are neighboring in the space of covariates by constructing a function, $f(x)$, that predicts z_i from x_i. Rather than update using 4.31 we can update our regression function by

$$\hat{F}(x) \leftarrow \hat{F}(x) + \rho f(x) \tag{4.33}$$

The remaining detail is the estimation of the step size ρ. We can again appeal to maximum likelihood to find an optimal value. Setting $g(\mu_i) = F(x_i) + \rho f(x_i)$ we can determine the maximum likelihood estimator for ρ. Note that now we have a simple generalized linear model where $f(x_i)$ is a covariate with coefficient ρ and $F(x_i)$ is an offset term. Using existing GLM software we can estimate the optimal step size.

Initialize $\hat{F}(x) = g(\bar{y})$ for all values of x.

For t in $1, \ldots, T$ do

1. Estimate the predictions as

$$\mu_i = g^{-1}(\hat{F}(x_i)). \tag{4.34}$$

2. Compute the working response

$$z_i \quad = \quad \frac{y_i - \mu_i}{a(\phi)V(\mu_i)g'(\mu_i)}. \tag{4.35}$$

   (We can actually drop the $a(\phi)$ since it is constant across all observations and step 5 scales the gradient optimally.)

3. Fit a regression model, $f(x)$, predicting z_i using x_i.

4. Estimate the optimal step size, ρ by fitting a GLM of the same class where $f(x_i)$ is the covariate with coefficient ρ and $\hat{F}(x_i)$ as an offset term.

5. With $\lambda \in (0, 1]$ as a learning rate parameter, update the boosted regressor as

$$\hat{F}(x) \leftarrow \hat{F}(x) + \lambda \rho f(x) \tag{4.36}$$

Return the final boosted regressor $\hat{F}(x)$.

Figure 4.3: The generalized boosted algorithm by gradient ascent

Lastly we can use 4.33 to update. Figure 4.3 summarizes the gradient ascent GBM algorithm.

We may have some covariates for which we do not desire boosted function estimates. That is, some covariates might be indicator variables or fixed linear effects. The weighted regression step for Fisher scoring or the gradient regression step in gradient ascent would involve only those covariates for which we want boosted estimates. For the remaining variables we can add another step to either the Fisher scoring or gradient ascent GBM algorithms which fits a GLM of the same class with the remainder of the covariates as predictors and the current $\hat{F}(x_i)$ as an offset term.

4.3 Boosting models for proportional hazards regression

Statistical methods for survival analysis model data in which the response variable is a lifetime or failure time (Cox 1972, Cox 1975, McCullagh and Nelder 1989, Kalbfleisch and Prentice 1980). Although these methods are mostly thought of in the context of survival times of patients in a medical experiment, they may also apply in other situations such as the time to failure of a mechanical component or the time until criminal recidivism. For these problems we simply wish to estimate the distribution of failure time given covariates. Censoring, the condition of an observation that is yet to fail at the end of the experiment, complicates the estimation process.

4.3.1 Proportional hazards regression

If $s(t|x)$ denotes the distribution of failure times for an observation with covariates x, then $S(t|x)$, its cdf, denotes the probability that an observation with covariates x has failed before time t. The hazard function,

$$h(t|x) = \frac{s(t|x)}{1 - S(t|x)}, \tag{4.37}$$

returns the "instantaneous risk" of failure, or $h(t|x)\delta t$ is the probability of failure in the next infinitesimally small interval of time given survival up to time t. Naturally, various choices of hazard functions imply different failure time distributions and vice-versa.

Proportional hazards models assume that the hazard function, $h(t|x)$, can be factored in to a component that explicitly depends on time and another, a regression function, that depends on the covariates.

$$h(t|x) = \lambda(t)e^{F(x)}, \tag{4.38}$$

We assume that the hazard ratio between two observations does not depend on time and is only a function of the difference in their regression functions. Common statistical practice assumes a linear model for $F(x)$, $F(x) = \boldsymbol{\beta^t x}$. As for the base-line hazard, $\lambda(t)$, we can assume that it takes on various parametric forms. Aitkin and Clayton (Aitkin and Clayton 1980) develop several survival models in this fashion. Assuming $\lambda(t) = \lambda$ results in an exponential distribution failure time model, $\lambda(t) \propto \alpha t^{\alpha-1}$ yields a Weibull distribution, and $\lambda(t) \propto \alpha e^{\alpha t}$ yields an extreme-value distribution (see table 4.2). Section 4.3.2 will discuss boosting the exponential failure time model and the extension to Weibull and extreme-value models.

When forming the likelihood, assuming a parametric form for $\lambda(t)$, each observation's contribution will depend on the failure indicator, δ_i. Observations for which we observe a failure time contribute $s(t|x)$. For censored observations we only know that they did not fail prior to time t and, therefore, they contribute $1 - S(t|x)$, the probability of that the survival time is greater than t. The log-likelihood for the regression function based on N possibly censored observations is

$$\begin{aligned}\ell(F|t,\delta,x) &= \sum_{i=1}^{N} \delta_i \log s(t_i|x_i) + (1-\delta_i)\log(1-S(t_i|x_i)) \\ &= \sum_{i=1}^{N} \delta_i \log \frac{s(t_i|x_i)}{1-S(t_i|x_i)} + \log(1-S(t_i|x_i)) \end{aligned} \tag{4.39}$$

Combining 4.37 and 4.38 we see that $S(t|x)$ satisfies the differential equation

$$\frac{S'(t|x)}{1-S(t|x)} = \lambda(t)e^{F(x)}. \tag{4.40}$$

and therefore

$$\log(1 - S(t|x)) \;=\; -\int_0^t \lambda(u)e^{F(x)}du. \tag{4.41}$$

Substituting 4.41 into 4.39 and rearranging we obtain the log-likelihood for the regression function $F(x)$.

$$\begin{aligned}\ell(F|t,\delta,x) &= \sum_{i=1}^{N} \delta_i \log h(t_i|x_i) - \int_0^{t_i} \lambda(u)e^{F(x_i)}du \\ &= \sum_{i=1}^{N} \delta_i[\log\lambda(t_i) + F(x_i)] - e^{F(x_i)}\int_0^{t_i} \lambda(u)du \end{aligned} \tag{4.42}$$

At this point we may choose among several parametric forms for the baseline hazard, $\lambda(t)$ and the regression function, $F(x)$. For example, assuming $\lambda(t) = \lambda$ and $F(x) = \boldsymbol{\beta}^{\mathbf{t}}\boldsymbol{x}$ results in the log-likelihood for the linear censored exponential regression model.

Clearly the drawback to this model formulation is the nuisance baseline hazard function. In practice we are usually more interested in good estimates of $F(x)$ than the entire hazard function. Cox (1972) and Cox (1975) developed a partial likelihood approach that avoids reference to the baseline hazard function in the estimating equations for $F(x)$. His strategy was to consider only those times points at which we observe a failure. Peto and Breslow, in their discussion of Cox's original paper, propose extensions and approximations when there are some tied failure times. In the infinitesimal time interval just prior to a particular failure time, t_i, several individuals might be at risk. This set of individuals, the "risk set" at t_i, are those who have not failed and have not been censored up this point in time. Given that one individual in the risk set failed at time t_i, the probability that a particular individual in the risk set was that failure is proportional to their hazard function. Since observation i was the observation that failed its contribution to the partial likelihood is

$$\frac{\lambda(t_i)e^{F(x_i)}}{\sum_{j=1}^{N} I(t_j \geq t_i)\lambda(t_i)e^{F(x_j)}} = \frac{e^{F(x_i)}}{\sum_{j=1}^{N} I(t_j \geq t_i)e^{F(x_j)}} \tag{4.43}$$

Conveniently the nuisance baseline hazard functions, $\lambda(t)$, cancel. The sum in the denominator in conjunction with the indicator function sums over the hazard functions of each of the observations in the risk set at time t_i. So this is simply the probability that observation i was selected from the risk set when the probability of selection is proportional to $e^{F(x_j)}$. At each failure time this same process of selection from a risk set occurs and so the partial likelihood is

$$PL(F|t,\delta,x) = \prod_{i=1}^{N} \left[\frac{e^{F(x_i)}}{\sum_{j=1}^{N} I(t_j \geq t_i) e^{F(x_j)}} \right]^{\delta_i} . \tag{4.44}$$

In practice we wish to choose $F(x)$ or parameter values for $F(x)$ to maximize the log-partial likelihood,

$$\log PL(F|t,\delta,x) = \sum_{i=1}^{N} \delta_i \left[F(x_i) - \log \left(\sum_{j=1}^{N} I(t_j \geq t_i) e^{F(x_j)} \right) \right] . \tag{4.45}$$

The log-partial likelihood takes into account only the order of the failure and censoring times.

In the next several sections I will discuss boosting proportional hazards models. I will start with a discussion of boosting parametric proportional hazards models before moving into various ways of boosting Cox's semi-parametric model.

4.3.2 Boosting parametric proportional hazards regression models

This section will discuss the application of boosting to proportional hazards models when we assume a parametric form for the baseline hazard function. In particular, I will deal with the three most common parametric survival models (exponential, Weibull, and extreme-value) but the method is general enough to admit an arbitrary failure time density function. Most importantly, this serves as a gentle introduction into boosting via partial likelihood and helps bridge the gap between the generalized boosted models and survival models.

On a population level we would like to have a regression function, $F(x)$, that maximizes the population form of equation 4.42.

$$\mathrm{E}_{t,\delta|x}[\ell(F|t,\delta,x)|x] \quad = \quad \mathrm{E}_{t,\delta|x}\left[\delta[\log\lambda(t) + F(x)] - e^{F(x)} \int_0^t \lambda(u)du \; |x\right] \tag{4.46}$$

Adding the constant $\delta \log \left(\int_0^t \lambda(u)du\right)$ and subtracting the constant $\delta \log \lambda(t)$ we can see that maximizing 4.46 is equivalent to maximizing

$$\mathrm{E}_{t,\delta|x}\left[\delta\left[F(x)+\log\left(\int_0^t \lambda(u)du\right)\right]-e^{F(x)+\log\int_0^t \lambda(u)du}\ |x\right]. \tag{4.47}$$

This is a canonical Poisson model for δ with regression function $F(x)$ and offset term $\log \int_0^t \lambda(u)du$. Since 4.47 is a canonical Poisson log-likelihood we know that $g(\mu) = \log \mu$ (from table 4.1) and that $g'(\mu)V(\mu) = 1$. We can almost directly apply the Fisher scoring algorithm from figure 4.2. The integrated hazard as an offset term is the only slight modification which causes the prediction in step 1 of the algorithm to be $\mu_i = e^{F(x_i)} \int_0^{t_i} \lambda(u)du$. The working response and weights then become

$$z_i = \frac{\delta_i}{e^{F(x_i)} \int_0^{t_i} \lambda(u)du} - 1 \tag{4.48}$$

$$w_i = e^{F(x_i)} \int_0^{t_i} \lambda(u)du. \tag{4.49}$$

The final update then proceeds as in step 5 of the algorithm. For a specific parametric family we simply need to submit the integrated baseline hazard to complete the algorithm. Table 4.2 presents the three most common parametric proportional hazards models. For the exponential model the regression function absorbs the hazard parameter λ. In both the Weibull and extreme-value cases there remains a free parameter α that requires estimation at the end of each boosting iteration. To estimate α in the Weibull case we simply need to fit a linear Poisson model for δ_i with predictor $\log t_i$ and offset term $F(x_i)$. The estimated coefficient of $\log t_i$ will be an MLE for α. The extreme-value case is similar with the exception of using t instead of $\log t$ as the predictor.

4.3.3 Cox's proportional hazards regression as Poisson regression

Section 4.3.2 developed boosting algorithms for use with parametric survival models and noted strong ties between Poisson regression and proportional hazards regression. This section discusses John Whitehead's (Whitehead 1980) method for using the

Table 4.2: Some parametric proportional hazards models

Distribution	$\lambda(t)$	$\int_0^{t_i} \lambda(u)du$
Exponential	λ	λt
Weibull	$\alpha t^{\alpha-1}$	t^α
Extreme-value	$\alpha e^{\alpha t}$	$e^{\alpha t}$

Poisson regression for fitting Cox's proportional hazards model. Section 4.3.4 then applies Whitehead's ideas directly to the generalized boosted model for obtaining boosted estimates of the regression function.

Whitehead (1980) showed that, at their maxima, Cox's partial likelihood is proportional to the Poisson likelihood of a transformation of the dataset. We can form an auxiliary dataset from the original survival dataset in the following fashion. Each observation in the auxiliary dataset will have the variables Y, h, and a vector of covariates, x. Consider the first failure time. Every observation in the risk set at the first failure time will enter the auxiliary dataset with $Y = 1$ if the observation failed at this time and $Y = 0$ otherwise, $h = 1$ – indicating the first failure time, and their covariates x. Processing the second failure, every observation in the risk set at the second failure time will enter the auxiliary dataset similarly with $Y = 1$ if the observation failed at this time and $Y = 0$ otherwise, $h = 2$ – indicating the second failure time, and their covariates x. The following example demonstrates this transformation.

Example 4.1 (Whitehead transformation)

Let t be the failure or censoring time, δ be the failure indicator, and x_1 and x_2 be the covariates.

Table 4.3: Original survival dataset

t	δ	x_1	x_2
0.1	1	5	1
0.3	0	3	1
0.7	0	7	3
1.1	1	1	2
1.5	1	6	4

For demonstration I will use the toy survival dataset in table 4.3.

We can form the auxiliary dataset in table 4.4 as previously described. Note that h indexes the failure times and, for each value of h, all the members of the risk set are represented. Since there are no tied observation in this example, $Y = 1$ for only one member of each risk set. This method easily incorporates Peto's method for handling tied observations by setting $Y = 1$ for more than one observation in the risk set.

Table 4.4: Auxiliary Poisson dataset

Y	h	x_1	x_2
1	1	5	1
0	1	3	1
0	1	7	3
0	1	1	2
0	1	6	4
1	2	1	2
0	2	6	4
1	3	6	4

Whitehead (1980) considered fitting the auxiliary model

$$Y_i \sim \text{Poisson}(\exp(\alpha_{h_i} + \boldsymbol{\beta^t x}_i)) \tag{4.50}$$

The estimates of the parameters α_h, associated with each failure time, are useful in predicting failure times. For the moment we will just assume they are nuisance parameters.

If we consider the observations in the risk set when $h = 1$, the first five rows of table 4.4, the Poisson log-likelihood is

$$\prod_{i^*=1}^{N^*} \left(\lambda_{i^*}^{y_{i^*}} e^{-\lambda_{i^*}}\right)^{I(h_{i^*}=1)} \tag{4.51}$$

where i^* indexes the N^* observations in the auxiliary dataset and the mean of each Poisson random variable is $\lambda_{i^*} = \exp(\alpha_{h_{i^*}} + \beta^t x_{i^*})$. Note that in our case for each value of h there is only one y that is non-zero, the one that is marked as a failure at the h^{th} failure time. And so the Poisson log-likelihood may be written as

$$\frac{\exp\left(\alpha_1 + \beta^t x_{i_1^*}\right)}{\exp\left(\sum_{i^*=1}^{N^*} I(h_{i^*} = 1) e^{\alpha_1 + \beta^t x_{i^*}}\right)}. \tag{4.52}$$

Here we have in the numerator the Poisson mean for the observation that failed. I will use i_h^* to indicate the index of the observation in the auxiliary dataset that failed at the h^{th} failure time. In the denominator we have the exponential of the sum of the Poisson means for the observations in the risk set.

From equation 4.51 we can derive that the maximum likelihood estimates for λ_{i^*} satisfy $\sum_{h_{i^*}=1} \hat{\lambda}_{i^*} = 1$. This implies that at the mle, the denominator of 4.52 equals e and that

$$e^{-\hat{\alpha}_1} = \sum_{i^*=1}^{N^*} I(h_{i^*} = 1) e^{\hat{\alpha}_1 + \hat{\beta}^t x_{i^*}}.$$

So, at its maximum the contribution to the Poisson log-likelihood of the observations in the auxiliary dataset that have $h = 1$ is

$$e^{-1} \frac{e^{\beta^t x_{i_1^*}}}{\sum_{i^*=1}^{N^*} I(h_{i^*} = 1) e^{\hat{\beta}^t x_{i^*}}}$$

which is beginning to look much like Cox's partial likelihood. In fact, incorporating the observations for the remaining values of h, the maximum of the Poisson log-likelihood is proportional to the Cox partial likelihood.

$$\prod_h e^{-1} \frac{e^{\hat{\beta}^t x_{i_h^*}}}{\sum_{i^*=1}^{N^*} I(h_{i^*} = h) e^{\hat{\beta}^t x_{i^*}}} \propto \prod_{i=1}^{N} \left(\frac{e^{\hat{\beta}^t x_i}}{\sum_{j=1}^{N} I(t_j \geq t_i) e^{\hat{\beta}^t x_j}} \right)^{\delta_i} \tag{4.53}$$

Therefore, fitting the Poisson model 4.50 will result in the same maximum likelihood estimates of β as directly fitting the Cox partial likelihood.

I will conclude this section by pointing out a few features and problems with this method. The main motivation behind this approach is to utilize the generalized boosted regression machinery already in place, the application of which I discuss in section 4.3.4. A relatively short program can transform the survival dataset into a suitable format for boosting via the Poisson model. This transformation increases the size of the dataset on the order of N^2. Although it depends on the dataset, I found that on average a small dataset of 1,000 observations can turn into one with 250,000 observations. If, however, the base regressor is a tree, a programming trick can greatly cut down on computation. Regression trees consider all possible splits in the dataset and select the one with the greatest likelihood. One can force the tree algorithm to search for splits among the 1,000 observations and use an on-the-fly Whitehead transformation of the dataset to compute the likelihood of the split. This removes the storage problem but still requires $O(N^2)$ operations.

An estimate of the baseline survivor function comes as a by-product of the Poisson regression framework. It is possible to use the estimates of α for the Breslow estimator (Breslow 1972) of the survivor function as

$$\exp\left(-\int_0^t \lambda(u)du\right) \approx \exp\left(-\sum_{t_h \leq t} \exp \hat{\alpha}_h\right). \tag{4.54}$$

Lastly, it is unclear in theory how this method works when making the transition to non-parametric regression. The above results indicate that when β is a finite vector of parameters the Poisson model is proportional to the Cox model *at the maximum*. At other locations the two models may have very different shape. Furthermore, in non-parametric regression we refrain from fully approaching the maximum of the likelihood (by smoothing or penalization) to prevent overfitting. We have no guarantees from the above results that dropping below the maximum still yields a good model. Empirically we found that this method still performed well on real datasets. In the next section we discuss those results. Later, in section 4.3.5 we will side step the Poisson model and propose a gradient ascent algorithm for maximizing the Cox model directly.

4.3.4 Boosting Cox's proportional hazards regression via boosted Poisson regression

After transforming the survival dataset into the auxiliary Poisson dataset via the Whitehead transformation, we have a dataset that we can model with a Poisson likelihood. For this section alone y_i will be the Poisson response, x_i the covariates, h_i the failure time indicator, and $\lambda(x_i, h_i)$ the mean of the Poisson distribution. The likelihood for our regression function and the α nuisance parameters becomes

$$\begin{aligned} \ell(F, \alpha) &= \sum_{i=1}^{n} y_i \log \lambda(x_i, h_i) - \lambda(x_i, h_i) \\ &= \sum_{i=1}^{n} y_i(\alpha_{h_i} + F(x_i)) - \exp(\alpha_{h_i} + F(x_i)). \end{aligned} \tag{4.55}$$

This is similar to the setup we had for parametric survival models in section 4.3.2. Again we can almost apply the generalized boosted model algorithm from figure 4.2. The algorithm requires slight modification since we have the additional α_h terms. Specifically, we modify the algorithm so that

$$\mu_i = \exp\left(\alpha_{h_i} + \hat{F}(x_i)\right) \tag{4.56}$$

$$z_i = \frac{y_i}{\mu_i} - 1 \tag{4.57}$$

$$w_i = \mu_i \tag{4.58}$$

In estimating this model we can alternate between making a boosting step and updating our estimates for α_h. After each update of the regression function we will also update α_h. Maximizing equation 4.55 for α_h accomplishes this.

$$\frac{\partial \ell}{\partial \alpha_h} = \sum_{i=1}^{n} I(h_i = h)\left[y_i - \exp(\alpha_h + F(x_i))\right] \tag{4.59}$$

Equation 4.59 implies that the updated estimate of $\hat{\alpha}_h$ is

$$\begin{aligned} \sum_{i=1}^{n} I(h_i = h) y_i &= \left[\sum_{i=1}^{n} I(h_i = h) e^{F(x_i)}\right] e^{\hat{\alpha}_h} \\ \hat{\alpha}_h &= \log \frac{\sum_{i=1}^{n} I(h_i = h) y_i}{\sum_{i=1}^{n} I(h_i = h) e^{F(x_i)}} \end{aligned} \tag{4.60}$$

The algorithm proposed here will find a regression function that maximizes the Poisson likelihood for the Whitehead transformation of the dataset. This transformation allows to reuse the same boosting algorithms presented earlier for boosting Poisson models. Although this conveniently takes advantage of the other algorithms, some delicate details remain as to whether using Whitehead's method can really obtain a decent estimate for the Cox model. The next section sidesteps this problem completely by directly maximizing Cox's partial likelihood by gradient ascent.

4.3.5 Boosting proportional hazards by partial likelihood

As previously mentioned our goal with boosting the proportional hazards model is to recover a function, $\hat{F}(x)$, that maximizes the expected value of the logarithm of the partial likelihood.

$$\mathrm{E} \log PL(F|t, \delta, x) = \mathrm{E}\left[\sum_{i=1}^{N} \delta_i \left[F(x_i) - \log\left(\sum_{j=1}^{N} I(t_j \geq t_i) e^{F(x_j)}\right)\right]\right] \tag{4.61}$$

This expectation is over all future observations that are potentially (or at least theoretically) observable. Practically this implies that on validation or test datasets we want our $\hat{F}(x)$ to yield a large log-partial likelihood.

In section 4.3.4 we created an auxiliary Poisson dataset and performed boosted Poisson regression. In this section we will select an F directly to maximize the expected log-partial likelihood.

For the moment we will assume that each observation in equation 4.61 can have its own F_i regardless of the covariates. Shortly we will utilize the covariate information to pull the F_i together. In this setting with sample data we wish to find N values $F_1, \cdots, F_N$ that maximize

$$\log PL(F|t,\delta,x) = \sum_{i=1}^{N} \delta_i \left[F_i - \log(\sum_{j=1}^{N} I(t_j \geq t_i) e^{F_j})\right]. \tag{4.62}$$

The gradient vector, given by $\frac{\partial}{\partial F_i} \log PL$, indicates the direction in which we would see the greatest increase locally in $\log PL$. Therefore, we would like to move all of the F_i in their gradient direction.

$$\begin{aligned}
\frac{\partial}{\partial F_i} \log PL &= \delta_i \left[1 - \frac{e^{F_i}}{\sum_{j=1}^{N} I(t_j \geq t_i) e^{F_j}}\right] \\
&\quad + \frac{\partial}{\partial F_i} \sum_{j \neq i} \delta_j \left[F_j - \log\left(\sum_{k=1}^{N} I(t_k \geq t_j) e^{F_k}\right)\right] &(4.63) \\
&= \delta_i \left[1 - \frac{e^{F_i}}{\sum_{j=1}^{N} I(t_j \geq t_i) e^{F_j}}\right] \\
&\quad - \sum_{j \neq i} \delta_j I(t_i > t_j) \frac{e^{F_i}}{\sum_{k=1}^{N} I(t_k \geq t_j) e^{F_k}} &(4.64) \\
&= \delta_i - \sum_{j=1}^{N} \delta_j I(t_i \geq t_j) \frac{e^{F_i}}{\sum_{k=1}^{N} I(t_k \geq t_j) e^{F_k}} &(4.65) \\
&= z_i
\end{aligned}$$

The first term in equation 4.63 comes from the derivative of the i^{th} term in the sum expressed in equation 4.61. The second term in equation 4.63 accounts for situations in which i^{th} observation is an element in the risk set at another failure time. In fact the partial derivative of that term will be non-zero in the event that t_j is a failure time and i's failure or censoring time occurs after t_j. The second term in equation 4.64 computes this partial derivative and all of 4.64 simplifies to 4.65.

Initializing $F_i = 0$ and letting z_i be the i^{th} element of the gradient vector, optimization of $\log PL$ would have us modify each F_i as

$$\mathbf{F} = \mathbf{F} + \rho \nabla \mathbf{F} \tag{4.66}$$

where ρ is the size of the step (constant across all i) in direction z_i that we will take.

At this point we reintroduce the covariates, x_i. Instead of allowing each of the observations to have their own F_i, which certainly would lead to rapid overfitting, we will smooth over neighboring x's. That is, we can compress the gradient information by construction a function of x, $f(x)$, that approximates the gradient, z. In Friedman's presentation of the gradient boosting machine (Friedman 1999a) he refers to this process as choosing the function from a parameterized class of functions that is most parallel with the gradient or "most highly correlated with z_i over the data distribution". Then rather than updated F_i, as in equation 4.66, we will update $F(x)$ as

$$\hat{F}(x) \leftarrow \hat{F}(x) + \rho f(x) \tag{4.67}$$

We will choose this function, $f(x)$, in 4.67 to minimize

$$\sum_{i=1}^{N} (z_i - f(x_i))^2. \tag{4.68}$$

We can use any standard regression procedure for this minimization step. In empirical work we will elect to use regression trees for their simplicity, utility for high dimensional problems with interactions, and speed.

Example 4.2 (Gradient step for the Cox model)

To demonstrate the gradient step approach, I let $F(x) = \frac{1}{2}\sin(3x_1 + 5x_1^2)$ be the underlying regression function and generated 100 survival times, t_i^{surv}, from an exponential distribution with mean $e^{-F(x_i)}$ where $x_i \sim U(0,1)$. Then for each observation

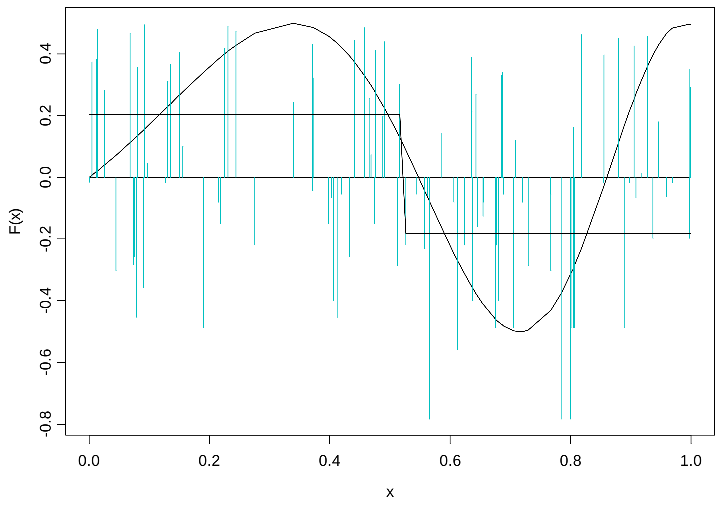

Figure 4.4: Diagram for example 4.2

I generated a censoring time, t_i^{cens}, from an exponential with mean 1. The final survival data consisted of survival times $t_i = \min(t_i^{\text{surv}}, t_i^{\text{cens}})$ and the failure indicator $\delta_i = I(t_i^{\text{surv}} \geq t_i^{\text{cens}})$.

Figure 4.4 show the true sinusoidal $F(x)$ and an initial guess for $F(x)$, the solid line constant at 0.0. The vertical lines indicate the direction and magnitude of the gradient computed at each of the 100 observations. This indicates for each x_i the direction and magnitude of change in the estimate for $F(x_i)$ that would locally increase the Cox partial likelihood. Fitting a one-split regression tree to the observed gradients using x as a predictor yields the step function. It captures at a very coarse level the gradient information and clearly makes a move in a promising direction toward capturing the true underlying regression function. Successive iterations proceed in the same fashion, further modifying the estimated $F(x)$ to maximize the Cox partial likelihood.

After obtaining an approximation to the gradient we need to choose the gradient step size, ρ. Naturally we want to select ρ to further maximize our log-partial likelihood. So given our function update, $f(x)$, the log-partial likelihood for ρ becomes

$$\log PL(\rho|F,f,t,\delta,x) = \sum_{i=1}^{N} \delta_i \left[F(x_i) + \rho f(x_i) - \log(\sum_{j=1}^{N} I(t_j \geq t_i) e^{F(x_j)+\rho f(x_j)})\right]. \quad (4.69)$$

Note that this is just the log-partial likelihood for a linear proportional hazards model where ρ is a coefficient for a continuous predictor, $f(x_i)$, with $F(x_i)$ as an offset term. We can derive a simple Newton algorithm for optimizing equation 4.69 with respect to ρ. The first and second derivatives of 4.69 with respect to ρ are derivable as follows.

Letting

$$w_j^{(i)} = I(t_j \geq t_i) e^{F(x_j)+\rho f(x_j)} \quad (4.70)$$

$$\frac{\partial}{\partial \rho} \log PL = \sum_{i=1}^{N} \delta_i \left[f(x_i) - \frac{\sum_{j=1}^{N} w_j^{(i)} f(x_j)}{\sum_{j=1}^{N} w_j^{(i)}} \right] \quad (4.71)$$

$$\frac{\partial^2}{\partial \rho^2} \log PL = -\sum_{i=1}^{N} \delta_i \left[\frac{\sum_{j=1}^{N} w_j^{(i)} f^2(x_j)}{\sum_{j=1}^{N} w_j^{(i)}} - \left(\frac{\sum_{j=1}^{N} w_j^{(i)} f(x_j)}{\sum_{j=1}^{N} w_j^{(i)}} \right)^2 \right] \quad (4.72)$$

Therefore, a Newton algorithm for updating ρ is

$$\rho \leftarrow \rho + \frac{\sum_{i=1}^{N} \delta_i \left(f(x_i) - \mathrm{E}_{w^{(i)}} f(x)\right)}{\sum_{i=1}^{N} \delta_i \mathrm{Var}_{w^{(i)}} f(x)} \quad (4.73)$$

Putting all these steps together we have a boosting algorithm for Cox's proportional hazards regression model shown in figure 4.5.

In the special case where the base regressor is tree-structured, we can further optimize the linear Cox model fitting step in step 3 and developed in the maximization of equation 4.69. Tree structured regressors (usually) fit piecewise constant functions. The tree predicts the same value for all observations that fall into a common leaf. We can retain the splits generated by the tree but improve upon the predictions offered

Initialize $\hat{F}(x) = 0$ for all values of x. For t in $1, \ldots, T$ do

1. Compute the working response.

$$z_i = \delta_i - \sum_{j=1}^{N} \delta_j I(t_i \geq t_j) \frac{e^{\hat{F}_i}}{\sum_{k=1}^{N} I(t_k \geq t_j) e^{\hat{F}_k}} \tag{4.74}$$

2. Construct a regressor, $f(\mathbf{x})$, predicting z_i from x_i.

3. Fit a linear proportional hazards model to the response (t, δ) with predictor $f(x_i)$, offset $F(x_i)$, and regression coefficient ρ.

4. Update the boosted estimate of $\hat{F}(x)$.

$$\hat{F}(x) = \hat{F}(x) + \lambda \rho f(x) \tag{4.75}$$

where λ is the pre-specified learning rate.

Figure 4.5: A Boosting algorithm for Cox's proportional hazards regression model

by each leaf. Let ℓ index the K leaves or terminal nodes of the tree $f(x)$ and $k(i)$ be the leaf index in which observation i fell. We can rewrite equation 4.69 as

$$\begin{aligned}
&\log PL(\rho|F, f, t, \delta, x) \\
&\qquad = \sum_{i=1}^{N} \delta_i \left[F(x_i) + \rho f(x_i) - \log\left(\sum_{j=1}^{N} I(t_j \geq t_i) e^{F(x_j)+\rho f(x_j)}\right)\right] \\
&\qquad = \sum_{i=1}^{N} \delta_i \left[F(x_i) + \rho f(k(i)) - \log\left(\sum_{j=1}^{N} I(t_j \geq t_i) e^{F(x_j)+\rho f(k(i))}\right)\right]. \qquad (4.76)
\end{aligned}$$

where $f(k)$ is the value that the k^{th} leaf assigns to its observations. With this structure there are only k possible values for $\rho f(k(i))$. We may be able to obtain better leaf estimates by maximizing 4.76 for k different ρ, one for each leaf.

$$\begin{aligned}
&\log PL(\rho_1, \cdots, \rho_k|F, f, t, \delta, x) = \\
&\qquad \sum_{i=1}^{N} \delta_i \left[F(x_i) + \rho_{k(i)} - \log\Big(\sum_{j=1}^{N} I(t_j \geq t_i) e^{F(x_j)+\rho_{k(i)}}\Big)\right]. \qquad (4.77)
\end{aligned}$$

Again this is the log-partial likelihood for a Cox regression model only this time $k(i)$ is a k level categorical or factor variable and $F(x_i)$ remains an offset. Although this might yield better estimates in the leaves of the regression tree, we now have a k dimensional optimization problem rather than a simple univariate one as first presented.

We can derive a Newton algorithm for maximizing 4.77 with respect to the vector of parameters $\boldsymbol{\rho}$. Let

$$p_i^{(k)} = \frac{\sum_{j=1}^{N} I(k(j) = k) I(t_j \geq t_i) e^{F(x_j)+\rho_k}}{\sum_{j=1}^{N} I(t_j \geq t_i) e^{F(x_j)+\rho_{k(j)}}}, \qquad (4.78)$$

which resembles a weighted estimate of the probability that a member of the i^{th} risk set falls in leaf k. We then can compute the gradient and the Hessian with respect to ρ_k of 4.77. The gradient is

$$g_k = \sum_{i=1}^{N} \delta_i \left[I(k(i) = k) - p_i^{(k)}\right]. \qquad (4.79)$$

The Hessian is a $k \times k$ matrix with diagonal elements

$$H_{mm} = \sum_{i=1}^{N} \delta_i p_i^{(m)}(1 - p_i^{(m)}) \tag{4.80}$$

and off diagonal elements

$$H_{mn} = -\sum_{i=1}^{N} \delta_i p_i^{(m)} p_i^{(n)}. \tag{4.81}$$

A Newton algorithm for obtaining maximum likelihood estimates for the leaf predictions is

$$\boldsymbol{\rho} \leftarrow \boldsymbol{\rho} - \boldsymbol{H}^{-1}\boldsymbol{g}. \tag{4.82}$$

4.3.6 Boosting proportional hazards directly

In section 4.3.4 and section 4.3.5 I developed boosting algorithms based on maximizing the Cox partial likelihood. The advantage of Cox's partial likelihood is that it avoids estimating the baseline hazard. However, we can develop a boosting algorithm that not only obtains a boosted estimate of the regression function but simultaneously obtains a boosted estimate of the baseline hazard.

As before, we would like to maximize

$$\ell(F, \lambda | t, \delta, x) \quad = \quad \sum_{i=1}^{N} \delta_i [\log \lambda(t_i) + F(x_i)] - e^{F(x_i)} \int_0^{t_i} \lambda(u) du \tag{4.83}$$

only now both the regression function, F, and the baseline hazard, λ, require boosted estimates. To improve upon the regression function estimate we can compute the gradient with respect to a specific $F(x_i)$.

$$\frac{\partial \ell(F, \lambda | t, \delta, x)}{\partial F(x_i)} = \delta_i F(x_i) - e^{F(x_i)} \int_0^{t_i} \lambda(u) du \tag{4.84}$$

The gradient gives the direction in which to move the regression function to further increase the likelihood. As before we can approximate the gradient by an arbitrary regression model, $f_F(x)$, and update the current boosted estimate by $\hat{F}(x) \leftarrow \hat{F}(x) + \rho_F f_F(x)$. We will select ρ_F shortly.

At the same time we would like to improve upon our estimate of the baseline hazard, $\lambda(t)$. Unfortunately we cannot compute the gradient of λ as easily as with ∇F since it is partially embedded in the integrated baseline hazard term, $\int_0^{t_i} \lambda(u)du$. The trapezoid rule is an integral approximation technique that is linear in λ. Assuming that the t_i are sorted in ascending order and using them as the nodes in the trapezoid approximation we obtain

$$\int_0^{t_i} \lambda(u)du \approx \frac{1}{2}(t_i - t_{i-1})\lambda(t_i) + \frac{1}{2}\sum_{j:t_j<t_i}(t_{j+1} - t_{j-1})\lambda(t_j) \tag{4.85}$$

where $t_0 = 0$. Now with this approximation we can compute the gradient with respect to $\lambda(t_i)$.

$$\frac{\partial \ell(F, \lambda|t, \delta, x)}{\partial \lambda(t_i)} \approx \frac{\delta_i}{\lambda(t_i)} + \frac{1}{2}e^{F(x_i)}(t_i - t_{i-1}) + \frac{1}{2}(t_{i+1} - t_{i-1})\sum_{j:t_j<t_i} e^{F(x_j)} \tag{4.86}$$

Similar to the regression function we can compress this gradient information by fitting a regression model, $f_\lambda(t)$, predicting 4.86 using only t_i as a predictor. We can update the current boosted estimate of $\lambda(t)$ by $\hat{\lambda}(t) \leftarrow \hat{\lambda}(t) + \rho_\lambda f_\lambda(t)$.

It remains to select the gradient step sizes ρ_F and ρ_λ. Again we can appeal to maximum likelihood to select values that maximize

$$\begin{aligned}\ell(\rho_F, \rho_\lambda|t, \delta, x) \quad = \quad & \sum_{i=1}^{N} \delta_i[\log(\lambda(t_i) + \rho_\lambda f_\lambda(t_i)) + F(x_i) + \rho_F f_F(x_i)] - \\ & e^{F(x_i)+\rho_F f_F(x_i)} \int_0^{t_i} \lambda(u) + \rho_\lambda f_\lambda(u)du \end{aligned} \tag{4.87}$$

If we add and subtract $\delta_i \int_0^{t_i} \lambda(u) + \rho_\lambda f_\lambda(u)du$ and take note of only the parts that deal with the regression function step size ρ_F we obtain

$$\begin{aligned}&\sum_{i=1}^{N} \delta_i \log\left(e^{F(x_i)+\rho_F f_F(x_i)} \int_0^{t_i} \lambda(u) + \rho_\lambda f_\lambda(u)du\right) - \\ &\qquad e^{F(x_i)+\rho_F f_F(x_i)} \int_0^{t_i} \lambda(u) + \rho_\lambda f_\lambda(u)du. \end{aligned} \tag{4.88}$$

Equation 4.88 is again a linear Poisson model with δ_i as the response variable, $f_F(x_i)$ as a covariate with coefficient ρ_F, and offset term $F(x_i) + \log\left(\int_0^{t_i} \lambda(u) + \rho_\lambda f_\lambda(u)du\right)$.

The baseline hazard gradient step size is not easily expressible as a coefficient of a GLM. A non-linear equation solver can estimate ρ_λ as the solution to

$$\sum_{i=1}^{N} \delta_i \frac{f_\lambda(t_i)}{\lambda(t_i) + \rho_\lambda f_\lambda(t_i)} - e^{F(x_i)+\rho_F f_\lambda(x_i)} \int_0^{t_i} f_\lambda(u)du = 0. \tag{4.89}$$

We then update the baseline hazard as $\lambda(t) \to \lambda(t) + \rho_\lambda f_\lambda(t)$. We can try to avoid solving equation 4.89 altogether by simply putting in a small positive fixed value for ρ_λ. It is unlikely that this would hurt performance too much and the worst it can do is require more iterations. Besides, iterations may be inexpensive in comparison to computing many non-linear solutions to 4.89. If a step solving 4.89 is included then it is necessary to iterate between 4.88 and 4.89 until both parameters converge.

4.4 Improving boosting methods using control of the learning rate, subsampling, robust regression, and decomposition for interpretation

This section explores the variations of the previous algorithms that have the potential to improve their predictive performance and interpretability. In particular, by controlling the optimization speed or learning rate, introducing low-variance regression methods, and applying ideas from robust regression we can produce non-parametric regression procedures with many desirable properties. As a by-product some of these modifications lead directly into implementations for learning from massive datasets. All these methods take advantage of the general form of boosting

$$\hat{F}(x) \leftarrow \hat{F}(x) + \mathrm{E}_w(z(y, \hat{F}(x))|x). \tag{4.90}$$

So far we have taken advantage of this form only by substituting in our favorite regression procedure for $\mathrm{E}_w(z|x)$. I will discuss some modifications to estimating $\mathrm{E}_w(z|x)$ that have the potential to improve our algorithm.

4.4.1 Decreasing the learning rate

As several authors have phrased slightly differently, "...boosting, whatever flavor, seldom seems to overfit, no matter how many terms are included in the additive expansion" (Friedman et al. 1998). This turns out to be patently false.

The following simple example illustrated in figure 4.6 demonstrates this.

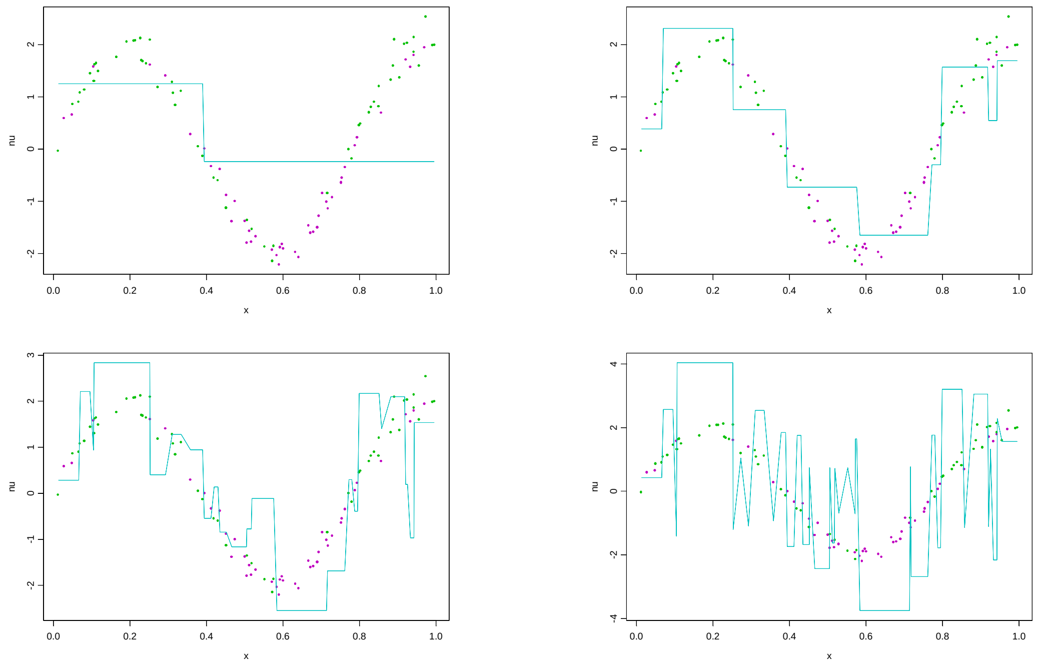

Figure 4.6: Iteration 1, 10, 20, 50 of the *LogitBoost* algorithm.

In this example I generated 100 observations with x_i coming from U$(0,1)$, $F(x) = 2(\sin(8x) + N(0, 0.1))$, and y_i from Bern$(p(x_i))$ where $p(x_i) = \frac{1}{1+e^{-F(x_i)}}$, the inverse logit transform. After 10 iterations we see that the boosting estimator is beginning to approximate the true underlying function. However, by iteration 20 and even more so at iteration 50 there is evidence of extreme overfitting. At this point the boosting procedure has carved up the x interval into regions that are nearly pure in terms of the training y_i's, clearly to the extent that the estimated $F(x)$ will not generalize well.

In estimating the curves shown in figure 4.6 the *LogitBoost* algorithm has been fairly aggressive (*AdaBoost* produces similar pictures). This aggressiveness is in the sense that it moved very quickly to a location in the space of regression functions that maximized the Bernoulli likelihood. In fact in only 25 moves it was showing signs of overfitting. Earlier work on boosting believed that boosting performed especially well in terms of generalization error because it achieved "slow learning". Breiman (1997) theorizes that the success of boosting algorithms depends on its ability to "skate around the inner rim [of the training error surface] instead of slogging to the bottom". He notes that his *arc-gv* algorithm reaches the bottom whereas algorithms such as *AdaBoost* and *arc-x4* tend to take large steps that "circle around the inner rim of the valley and do not reach the bottom". He wonders how far one should allow the boosting algorithm to descend in this error surface.

Here I propose the introduction of a learning rate parameter $\lambda \in (0, 1]$. In the update step of any boosting algorithm we can involve the learning rate to dampen the proposed move.

$$\hat{F}(x) \leftarrow \hat{F}(x) + \lambda \mathrm{E}_w(z(y, \hat{F}(x))|x). \tag{4.91}$$

By multiplying the gradient step by λ as in equation 4.91 we have control on the rate at which the boosting algorithm descends the error surface (or ascends the likelihood surface). When $\lambda = 1$ we return to performing full gradient steps. Recent work by Friedman (Friedman 1999a) relates the learning rate to regularization through shrinkage. Copas (1983b) argues that to improve predictive performance of regression models one should predict using models of the form

$$\bar{y} + \lambda(\hat{F}(x) - \bar{y}). \tag{4.92}$$

When λ is 1 the prediction is entirely from $F(x)$. When λ is 0 the prediction is entirely based on $\bar{y}$. In between 0 and 1 the predictor is a mixture of the regression model with covariates x and the constant model. He argues that setting $\lambda < 1$ gives a better fit to validation data. On each iteration of the boosting algorithm we can invoke a shrinkage of the regression estimate as in 4.91.

I experimented with creating optimal learning schedules so that λ_t depends on the iteration number. The motivation being that the first few boosting steps should be straightforward. While the boosting proceeds we might consider cooling off the learning process, eventually hitting 0. However, I found no learning schedule that could perform better than constant slow learning.

The optimal number of iterations, T, and the learning rate, λ, depend on each other. In practice I set T to be as large as possible and then select λ by cross-validation. When λ is as small as possible performance seems to be at its greatest. Figure 4.7 shows the *LogitBoost* after 3000 iterations with $\lambda = 0.01$ on the same example described earlier in this section. Clearly it performs much better than the estimator with the larger learning rate. Figure 4.8 shows the increase in likelihood across the iterations. After about 1000 iterations it has surpassed the GAM model (the horizontal line) and has reached a fairly long plateau. I have found these plateaus to be commonplace features of boosting algorithms especially with very small learning rates. This implies that boosting with slow learning is fairly robust to selection of the number of iterations and so takes much longer to overfit.

These slower learning rates do not necessarily scale the number of optimal iterations. That is, if when $\lambda = 1.0$ and the optimal T is 100 iterations, does *not* necessarily imply that when $\lambda = 0.1$ the optimal T is 1000 iterations. That is not even the most critical feature of the learning rate reduction. Figure 4.9 shows the progression for decreasing values of the learning rate. In this case the optimal T occur at roughly 10, 200, and 1500 for the three values of λ. Most notably, however, with the slower learning rates we are able to achieve a greater improvement, although with a decreasing marginal return on computational investment. This figure shows that the validation likelihood increases and the sensitivity to T decreases.

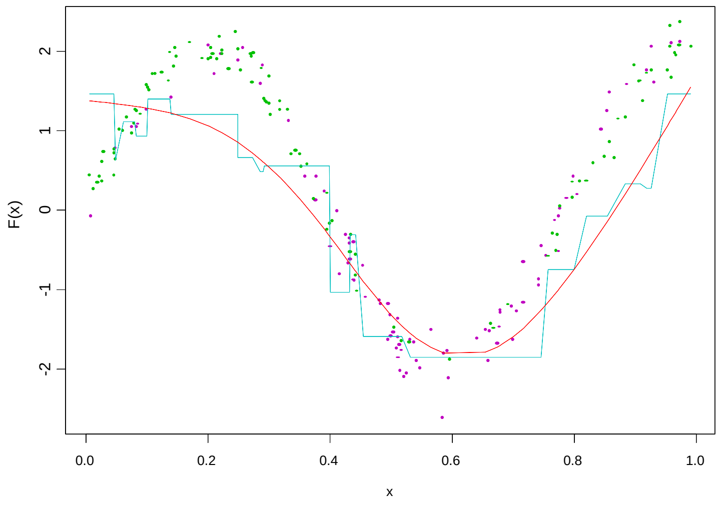

Figure 4.7: $\hat{F}(x)$ after 3000 iterations with the learning rate $\lambda = 0.01$. The smooth curve is the GAM fit and the step function is the boosted estimate.

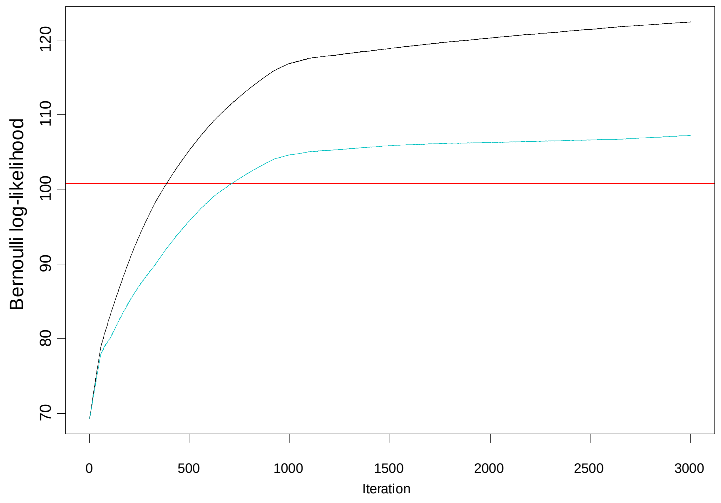

Figure 4.8: Associated Bernoulli log-likelihood. The horizontal line is GAM, the upper curve is the training likelihood, and lower curve is the validation likelihood.

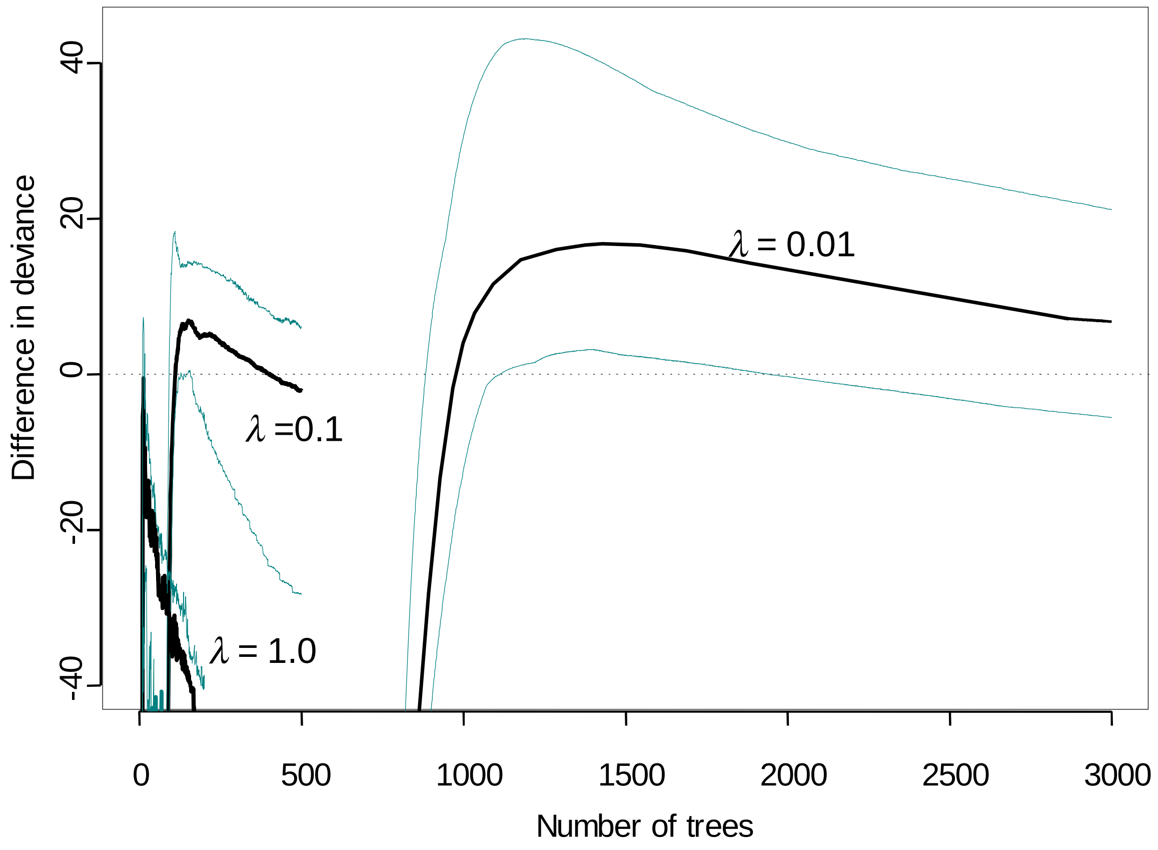

Figure 4.9: Effect of learning rate on exponential survival models: Improvement with respect to the linear exponential survival model for three values of λ.

4.4.2 Variance reduction using bagging and sub-sampling

Breiman (1996a) introduced bagging (*b*ootstrap *ag*gregating) as a variance reduction method for unstable regression and classification models. Bagging proceeds by creating B bootstrap replicates of the dataset and fitting a model to each. The final bagged regression model is the average of the predictions from the B models. With weak theoretical arguments and substantial empirical ones he shows that bootstrapping simulates an endless stream of new data adequately enough so that averaging reduces prediction variance. Friedman and Hall (1999) note that bagging replaces model components that are non-linear in the data with their expected value while leaving the linear components unaffected. By this argument they conclude that estimators like the sample mean and linear regression would be unaffected by bagging while bagged trees and neural networks might achieve variance reduction. They also theorize that sub-sampling half of the dataset without replacement on each iteration is approximately equivalent to bagging, sampling the entire dataset with replacement, in terms of the degree of variance reduction, but also indicate that sampling fractions less than half could reduce the variance further.

Extending the bagging idea, Breiman (1999) developed adaptive bagging that has strong connections to boosting methodology. Discussed primarily in the regression setting, he proposes fitting a bagged regression model to a dataset, estimating the bias using an "out-of-bag" method (Breiman 1996c), and fitting a new bagged regressor to the bias. Repeating this process of estimating the current bias yields his adaptive bagging algorithm. When boosting the Gaussian case, if on each step we estimate $\mathrm{E}_w(z|x)$ using B predictors built on bootstrap replicates of the dataset and estimate $\hat{F}(x)$ for the training dataset using the out-of-bag technique, then we have adaptive bagging exactly. Generalizing boosting as I have shown here, opens up exponential family and proportional hazards regression models to Breiman's adaptive bagging methodology.

To date, however, boosting proceeds in a slightly different fashion from adaptive bagging. It fits the residuals on each iteration but lacks the perturbation introduced by bootstrapping and the out-of-bag bias estimation. Breiman (1997) proposed introducing a weighted bootstrap step (sampling with replacement according to the weights) into boosting algorithms and conjectured that "randomizing the path gets *arc-gv* away from the bottom of the [error surface] so it skates around the inner rim". Friedman (1999b), inspired by Breiman (1999), proposed the stochastic gradient boosting algorithm that simply samples uniformly without replacement from the dataset before estimating the next gradient step. He found that this additional step greatly improved performance.

Sub-sampling techniques also lead directly onto applications for massive datasets. Computation time obviously decreases when we are able to use smaller sample sizes on each iteration, quadratically so for proportional hazards models. Breiman (1996d) suggests that building a sequence of models to "bites" of the datasets and "pasting" them together results in little loss and often some improvement. When the gradient steps involve a non-uniform weighting on the observations we can further decrease the sample size without interfering with the model's performance. After several iterations some of the observations are likely to have small weights, indicating that the model fits them well. At this stage they can be trimmed from the model fit stage. Often we can trim a large fraction of observations that have weights that sum to less than one observation. Although these observations may be trimmed from the model fitting stage they do need reweighting after each iteration in case at some point the model begins to predict them poorly.

Revisiting the boosting update step, we now have a learning rate parameter and we estimate $\mathrm{E}_w(z(y,\hat{F}(x))|x)$ using a random subsample of the dataset.

$$\hat{F}(x) \leftarrow \hat{F}(x) + \lambda \mathrm{E}_w(z(y,\hat{F}(x))|x). \tag{4.93}$$

In the special case where we use regression trees to estimate $\mathrm{E}_w(z(y,\hat{F}(x))|x)$ the sub-sampling increases the number of possible split points. This permits the boosted estimate to achieve smoother fits to the data.

4.4.3 Robust regression

The presence of outliers can seriously degrade the performance of many regression procedures. When the error distribution departs from the assumed distribution outliers may be common. Under certain loss functions, like squared error loss, the presence or absence of one particular observation may produce drastically different models. Well-designed robust regression methods are efficient when there are no outliers and are resistant to fitting outliers too closely when they are present. Rather than use squared error loss, robust location estimators minimize functions like least absolute deviation or Huber's loss (Huber 1964).

To incorporate robust methods we simply modify the estimate $\mathrm{E}_w(z(y, \hat{F})|x)$ to be a minimizer of a loss function other than the usual squared error loss.

$$\hat{F}(x) \leftarrow \hat{F}(x) + \lambda \mathrm{E}_w(z(y, \hat{F}(x))|x). \tag{4.94}$$

Friedman (1999a) develops an entire boosting algorithm for obtaining regression functions that minimize least absolute deviation and M-regression. In contrast here, I am only interested in a *robust gradient step* that would keep the entire process of boosted estimation from being attracted to outlying observations.

4.4.4 Interpretability

In his work on MARS, Friedman (1991) noted that certain function approximation methods are decomposable in terms of a "functional ANOVA decomposition". That is a function is decomposable as

$$F(x) = \sum_j f_j(x_j) + \sum_{jk} f_{jk}(x_j, x_k) + \sum_{jk\ell} f_{jk\ell}(x_j, x_k, x_\ell) + \cdots. \tag{4.95}$$

Friedman et al. (1998) note that this applies directly to boosting trees. Regression stumps (one split decision trees) depend on only one variable and fall into the first term of 4.95. Trees with two splits fall into the second term of 4.95 and so on. By restricting the depth of the trees produced on each boosting iteration we can control the order of approximation. Often additive components are sufficient to approximate a multivariate function well, generalized additive models, the naïve Bayes classifier, and boosted stumps for example. When the approximation is restricted to a first order we can also produce plots of x_j versus $f_j(x_j)$ to demonstrate how changes in x_j might affect changes in the response variable.

Friedman (1999a) also develops an extension of a variable's "relative influence" (Breiman et al. 1984) for boosted estimates. For tree based methods the approximate relative influence of a variable x_j is

$$\hat{J}_j^2 = \sum_{\text{splits on } x_j} I_t^2 \tag{4.96}$$

where I_t^2 is the empirical improvement by splitting on x_j at that point. Friedman's extension to boosted models is to average the relative influence of variable x_j across all the trees generated by the boosting algorithm.

4.5 Performance

We have seen so far that boosting is applicable to a large class of models, the exponential family models and proportional hazards models, and that we can incorporate into the boosting process many of the statistical properties that we desire in function estimators, robustness, stability, and efficiency in computation. In this section I discuss how well boosting performs in practice on real and simulated datasets. Having presented several boosting algorithms for models in standard statistical practice and discussing further variations on them, the real test remains as to whether they should be adopted for common use.

Table 4.5: Misclassification rates comparison for

	CART	*AdaBoost*	*LogitBoost*
Breast	4.5%	4.0%	2.9%
Ion	7.6%	6.8%	7.1%
Glass	40.0%	25.7%	26.6%
Sonar	59.6%	20.2%	20.2%
Waveform	36.4%	19.5%	20.6%

Friedman, Hastie, and Tibshirani (1998) first proposed, *LogitBoost*, a boosting algorithm for the canonical Bernoulli model or logistic regression and compared their results with the *AdaBoost* classifier. As shown in table 4.5, they found that their likelihood based *LogitBoost* algorithm was nearly equivalent in performance to *AdaBoost* and clearly an improvement on a single decision tree. The CART algorithm used the cross-validation for pruning and the two boosting algorithms iteratively built $T = 200$ one-split CART models (stumps).

Friedman (1999a) discussed boosting the Gaussian model as well as contaminated Gaussian error distributions and made comparisons with MARS. Unlike the tree-based gradient boosting methods, MARS produces a continuous function approximation. Despite the discontinuous approximation, he found that on average gradient boosted trees did slightly better than MARS in terms of absolute error and 30% better in terms of root mean squared error. Furthermore, the gradient methods produced lower variance approximations.

4.5.1 Additive Poisson regression

In this section I will discuss the performance of the boosted Poisson model, which has not yet made an appearance in any of the boosting literature. The main competitor to the boosted Poisson model is the Poisson GAM model to which all comparisons will be made.

To test this method I proposed a bivariate sinusoidal function 4.97 to be the true underlying regression function. It is additive in the predictors but has variable curvature.

$$F(x_1, x_2) = 2\sin(3x_1 + 5x_1^2) - 2\sin(3(x_2 + 0.1) + 5(x_2 + 0.1)^2) \tag{4.97}$$

I generated 1,000 values from a uniform distribution on (0,1) for each of x_1 and x_2. I then generated the response variable as $y_i \sim \text{Poisson}(e^{F(x_{i1}, x_{i2})})$. The goal of the boosting algorithm is to use these observations to estimate $F(x_1, x_2)$ and maximize a validation log-likelihood.

I used the first 500 observations as a training set and the remaining 500 observations as a test set. With the learning rate set at $\lambda = 0.1$, I ran the boosting algorithm for $T = 1,000$ iterations with regression stumps as the base regressor using only the training set. I also varied the sub-sampling percentage for values between 10% and 100% (no sub-sampling) at intervals of 10%. So for a particular training set I produced one GAM and 10 boosted Poisson models. To note boosting's performance I recorded the difference between the log-likelihood of each of the boosted Poisson models and the one Poisson GAM. This maintains the improvement on GAM in terms of validation log-likelihood for each sub-sampling rate. I repeated this experiment for 20 different training and test datasets.

Figure 4.10 displays the results of this experiment. For each sub-sampling rate I computed the median of boosting's improvement on GAM in terms of test set log-likelihood. The figure also shows the estimated 10% and 90% quantiles. There are two important conclusions to draw from this figure. The first obvious one is that boosting almost always outperforms the Poisson GAM. In only one of the trials involving a sub-sampling rate of 10% did GAM outperform boosting. The second important observation from figure 4.10 is that the performance is strong when the sub-sampling percentage is 20% or higher. The actual empirical maximum occurs at

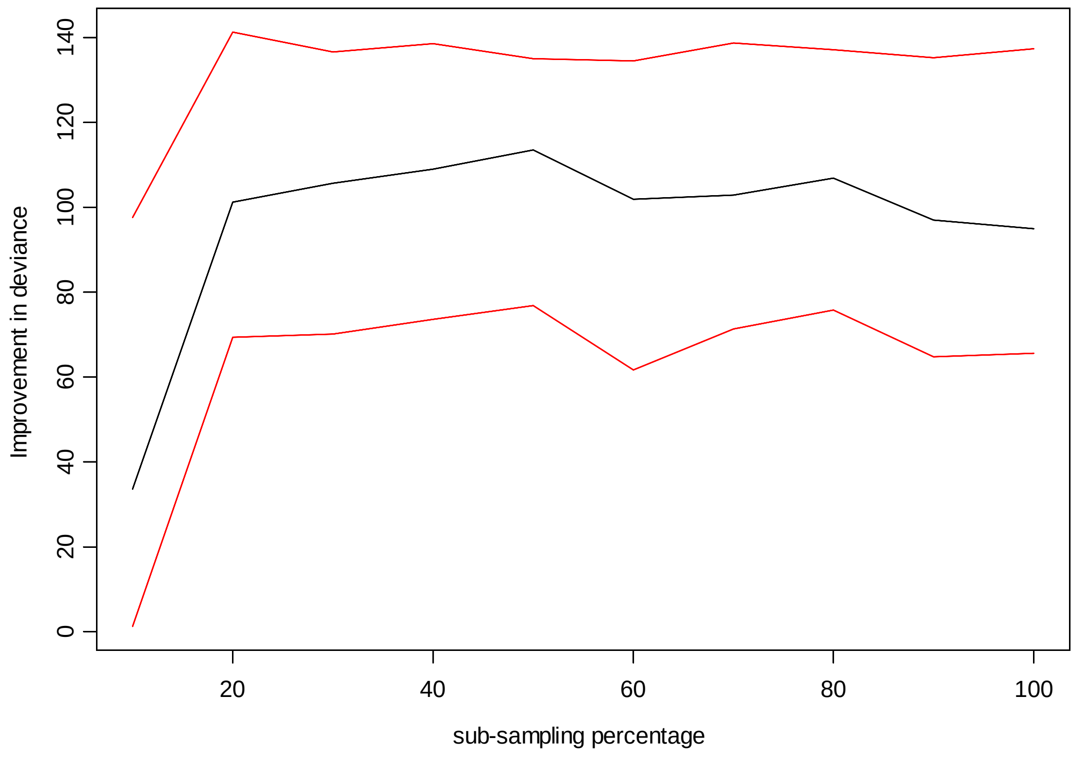

Figure 4.10: Boosting's absolute improvement over GAM for an additive Poisson model as a function of sub-sampling percentage. The middle curve indicates the median of boosting's improvement in validation log-likelihood compared to the equivalent GAM model over 20 repetitions. The outer curves indicate the 10% and 90% improvement quantiles.

50%, but that maximum does not exceed the performance on the interval (20%,100%) by very much. This indicates that boosting really does not need to look at all the data on each iteration to make reasonable steps toward function optimization. This experiment used random sampling to select the subset of the dataset to use on each boosting step. Sampling in massive datasets is not necessarily a trivial matter (Brutlag and Richardson 1999). However, it may be that a deterministic sampling scheme of convenience may be a viable substitute to random sampling, still maintaining good predictive performance and gaining further computational performance.

I repeated this experiment in a second trial, this time testing how boosting handles noise in the regression function and irrelevant predictors. I used the same simulation procedure as before with the following changes. I added a Gaussian noise component to $F(x_1, x_2)$ tuned so that the true underlying function explained about 93% of the variability ($\sigma = 0.5$). I also gave to both GAM and the boosted Poisson algorithm two additional predictor variables x_3 and x_4 (both generated as uniform on the unit interval), neither of which have any influence on the response variable.

Figure 4.11 shows the results for this second experiment. Although the boosted Poisson model still outperforms GAM most of the time, the addition of noise and irrelevant variables reduces the improvement. Furthermore, the figure indicates that the model fitting demands more data to be competitive. A sub-sampling rate of less than 30% seriously degrades the performance.

4.5.2 Cox proportional hazards

For a real data set example to examine the performance of boosting Cox's proportional hazards model, I turn to a clinical trial for testing the drug DPCA for the treatment of primary biliary cirrhosis of the liver (PBC). This dataset has been the subject of several modern data analyses (Dickson et al. 1989, Grambsch et al. 1989, Markus et al. 1989, Fleming and Harrington 1991). Raftery, Madigan, and Volinsky (1996) demonstrated the use of Bayesian model averaging on the PBC dataset and found

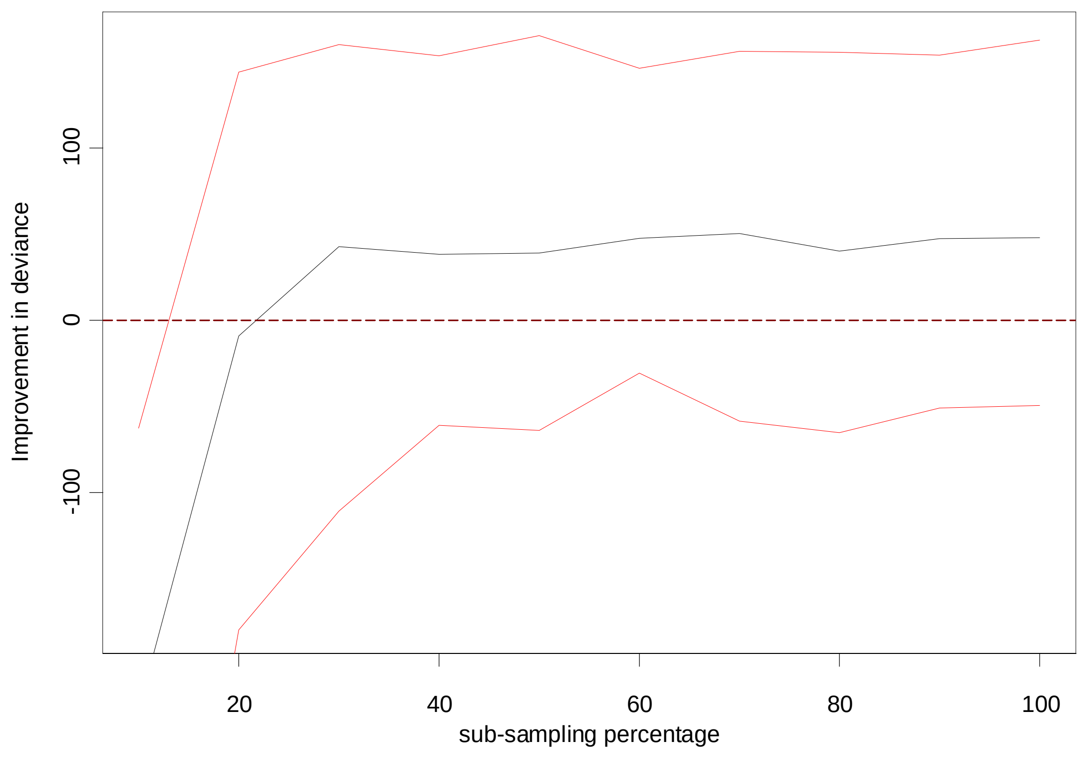

Figure 4.11: Boosting's absolute improvement over GAM for a noisy additive Poisson model with irrelevant predictors as a function of sub-sampling percentage. The middle curve indicates the median of boosting's improvement in validation log-likelihood compared to the equivalent GAM model over 20 repetitions. The outer curves indicate the 10% and 90% improvement quantiles.

that accounting for model uncertainty improves predictive performance. Boosting, on the other hand, does not mix models as Bayesian model averaging does but rather uses several models to estimate the non-linear effects relating the predictor variables to survival times.

The data consist of 310 patients with complete observations on the predictors. Of these, 124 patients died during the study and the remaining 186 were censored observations. Of the eight variables Raftery et al. (1996) considered, I selected the six real valued variables for use in the model.

age	age in years
bili	serum bilirubin (mg/dl)
albumin	albumin in g/dl
copper	urine copper in μg/day
sgot	SGOT in U/day (aspartate aminotransferase)
protime	prothrombin time in seconds

I tested this method by comparing the out-of-sample predictive performance of the linear Cox model to the Cox model boosted by gradient ascent. To judge the out-of-sample predictive performance of the two models, I trained each on half of the observations, reserving the rest for a test set. I used regression stumps as the base regressor so that the estimate only captured the main additive effects. Setting the learning rate to $\lambda = 0.00001$ and the subsampling rate to 10%, I ran the algorithm for 70,000 iterations. Figure 4.12 shows the value of the validation log-likelihood as the algorithm proceeds. We see that after 40,000 iterations (with this extremely low learning rate) the boosted estimate has surpassed the linear Cox model. Furthermore, for the next 30,000 iterations the boosted version continues to improve at a diminishing rate of return. Even though the model fitting used many iterations, it takes about one minute to complete the iterations.

Since the base regressors were one-split decision trees, we can investigate the estimates of the six main additive effects. Figure 4.13 shows the boosted estimates

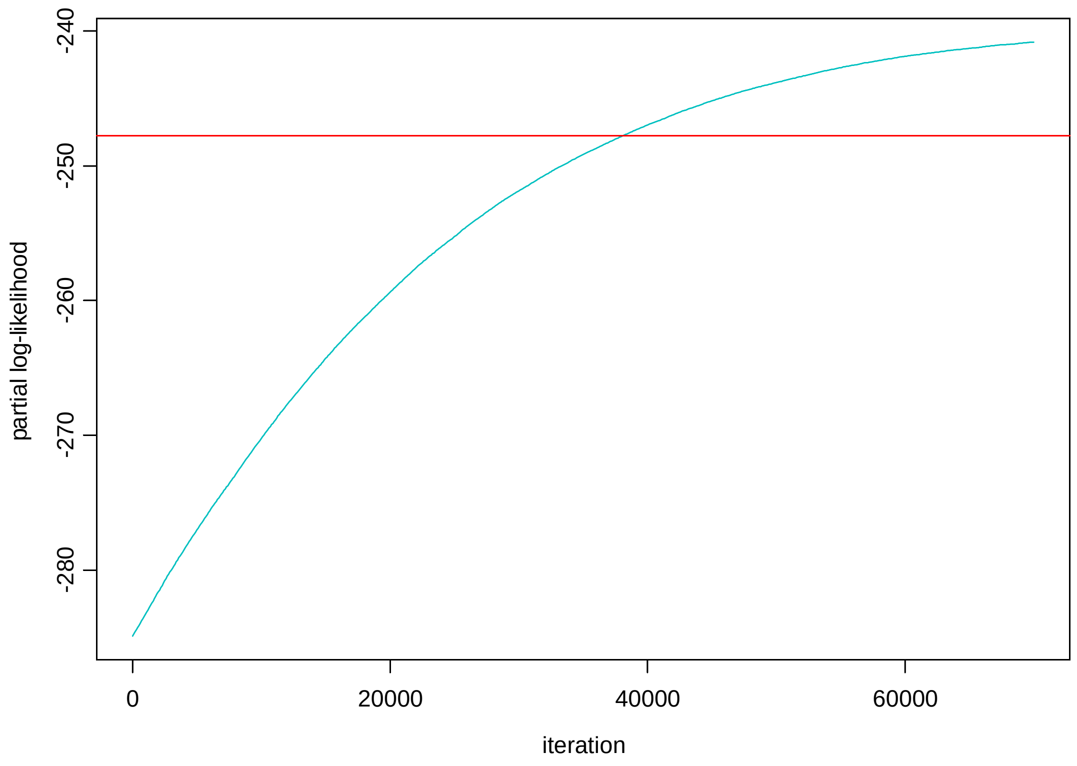

Figure 4.12: Partial log-likelihood for PBC data. The upward trending curve indicates the partial log-likelihood for test set data as the number of boosting iterations increases. The horizontal line indicates the partial likelihood from the linear Cox model.

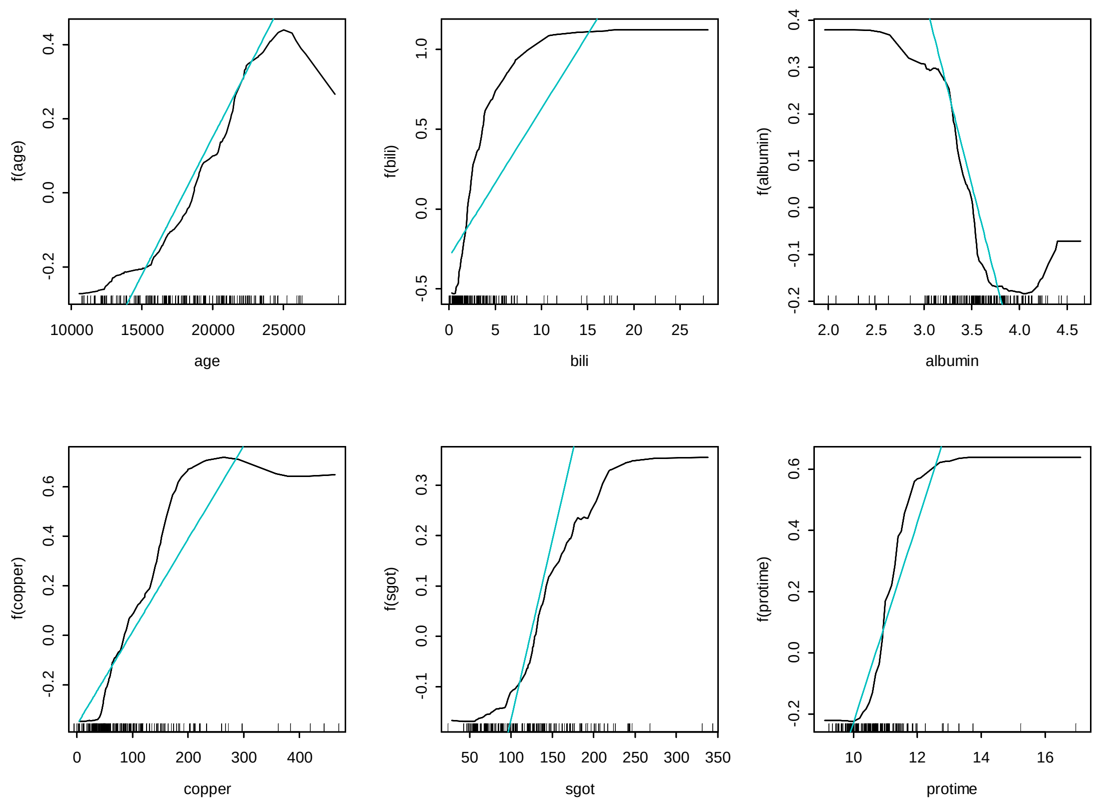

Figure 4.13: Boosted estimates of the main effects of the PBC data. The curves are the boosted estimates of the functions and the superimposed lines are the estimates from the linear model

along with those based on the linear model. Every variable shows evidence of either a threshold effect, a saturation effect, or both. Simple linear models clearly can never capture threshold and saturation effects. Although in the region where most of the data points are concentrated, for most of the variables (except bili) the underlying regressor is nearly linear. For this reason we would expect the linear model to function reasonably well. However, for a patient with a more extreme value for any of the six variables the boosted model is far superior. Even though most applications of survival models are not concerned with survival prediction but rather estimation of effects, when effects depart substantially from linearity, as in the PBC dataset, accurate survival prediction for all patients depends on a non-linear estimation procedure like boosting.

4.6 Discussion

The trend toward model mixing had a resurgence in economics (Bates and Granger 1969), has increased in the machine learning community, and is partially accepted in the statistics community. Researchers in these fields proposed techniques like bagging (Breiman 1996a), Bayesian model averaging (Madigan et al. 1996, Hoeting et al. 1999), stacking (Wolpert 1992), and bumping (Tibshirani and Knight 1995) showing that their methods solved problems associated with dependence on one selected model. At its first introduction computational learning theorists believed that boosting also fit in this class since it built several models on different weightings of the dataset and merged them together. When boosting simple base classifiers, predictive performance on many of the classic benchmark datasets increased. Researchers believed these results implied that boosting was now the best of this class of algorithms.

Research, including this work, is now showing that boosting represents a new class of learning algorithms that Friedman correctly named "gradient machines". As all the extensions described here show, boosting iteratively fits some form of the residual, regions of the sample space where the model's predictions are missing the target data. Although the various boosting classification procedures (*AdaBoost*, *LogitBoost*, etc.) altered the loss function that the algorithm optimized, the subsequent iteration simply fit the training residuals for which the current regression function estimate did not yet account. This being the case the original class of model mixing procedures are not in competition with boosting but rather can coexist inside or outside a boosting algorithm. That is, one could average across boosted estimates or perform bagging and bumping within a boosting iteration.

In the last few years boosting has improved the performance of many classifiers on real datasets. I have been able to import this methodology into statistics by taking the perspective that boosting algorithms are functional optimization algorithms. Using various optimization techniques I have produced classes of algorithms for boosting models that are a part of standard statistical practice, the exponential family and proportional hazards regression models.

Several authors have already observed improvement in predictive performance for logistic, normal, and robust regression. Here I have shown that the same ideas can improve predictive performance for Poisson (compared with the Poisson generalized additive model) and Cox proportional hazards models (compared to the linear Cox model).

Software written in C++ with an optional S-plus interface is available from the author for the boosted exponential family models and the Cox proportional hazards model. As research evolves in this area, particularly incorporating variations on the gradient steps, this software will need to incorporate these enhancements. These algorithms show great promise for genuine massive dataset analysis through robust regression, weight trimming, and gradient stepping using disjoint blocks of data. Although the algorithm's derivation exposes stages at which these improvements could be made, the software implementation so far does not take advantage of all these features.

To summarize, in this chapter I have

- developed a general boosting algorithm for exponential family models based on Fisher scoring,
- showed that the *LogitBoost* algorithm of Friedman, Hastie, and Tibshirani (1998) falls in this class,
- derived a boosting algorithm for exponential family models based on gradient ascent, which can use existing software for the related linear model,
- produced boosting algorithms for parametric proportional hazards models,
- extended the boosted Poisson model by Fisher scoring as an approximation to boosting the Cox model,
- extended the gradient ascent approach to boost the Cox proportional hazards model, an algorithm which can utilize software for fitting the linear Cox model,
- discussed improving existing boosting algorithms by incorporating control over the rate of learning, variance reduction, robust methods, interpretability, and data reduction for computational performance,
- evaluated the boosted Poisson model on simulated data, and
- evaluated the boosted Cox model on a real dataset for primary biliary cirrhosis.

This is a large step in demonstrating the wide applicability of boosting methods. Boosting has the potential to work its way into all the other regression models that remain at large.

Chapter 5

BAYESIAN METHODS FOR MASSIVE DATASETS

The advent of the massive dataset and the expansion of the field of data mining has created the need to produce statistically sound methods that scale to these large problems. Although statistics in the past has dealt with extracting maximum information from a small dataset, these new problems require methods that can compress the massive dataset in a reasonable amount of time into something humanly understandable.

In this chapter I will propose one method that permits Bayesian inference in massive datasets for a large class of models. I will also outline some ideas for performing approximate inference based on "likelihood clustering".

5.1 Statistics and data mining

The rapid increase in automated information gathering from sources such as the checkout counter, the World Wide Web, and orbiting satellites has far outpaced the development of tools to analyze data flowing in such quantities. The field of Knowledge Discovery and Data mining (KDD) grew primarily out of computer science to meet the growing need. Even though statisticians have been at the forefront of data analysis for over a century, data mining grew with little interaction from the statistical community. The datasets are massive, high-dimensional, often retrospective, and rarely clean. Nevertheless, the market demand exists for new methods of data management and new sets of algorithms to extract accurate and ultimately useful information from these massive stores of data.

The need for proper statistical analysis has not gone unnoticed in the data mining community. Statistical concepts such as latent variables, spurious correlation, and problems involving model search and selection have appeared in widely noted data mining literature (Elder and Pregibon 1996, Glymour, Madigan, Pregibon, and Smyth 1997). However, algorithms, model fitting methods that actually work on massive datasets, have been slow to appear. Too many of these algorithms begin by a "load data into memory" step. The vision for an ideal data mining algorithm is one that requires a single scan of the dataset to produce the final result. For all but the simplest of models in which sufficient statistics are easily computable this goal is far from being obtained.

If the most severe penalty comes when requesting data, algorithms might exist that only use small manageable portions of the dataset at any one time. A second stage might then perform a sequential scan of the remaining observations, modifying the model fit with each additional observation. In this chapter, I introduce a Bayesian framework that may be a starting point for such algorithms.

5.2 An importance sampling algorithm for Bayesian inference in massive datasets

In this section I introduce an algorithm based on importance sampling to facilitate Monte Carlo simulations when the posterior distribution conditions on a massive dataset. The method uses a posterior distribution conditioned on a smaller, more manageable partition of the dataset as a sampling distribution. With this choice the importance weight function is simple for any readily computable likelihood function. Incorporating the entire dataset requires only one scan of the remaining observations. Derived from an adaptive importance sampling algorithm, I propose predictive weight trimming to gain further sampling efficiency.

One might first wonder if Monte Carlo techniques for a full Bayesian analysis are necessary. Except for the simplest of models and regardless of the style of inference, estimation algorithms almost always require repeated scans of the dataset. We know that for well-behaved likelihoods and priors, the posterior distribution converges to a multivariate normal (Le Cam and Yang 1990, DeGroot 1970). For large but finite samples this approximation works rather well on marginal distributions and lower dimensional conditional distributions but does not always provide an accurate approximation to the full joint distribution (Gelman, Carlin, Stern, and Rubin 1995). The normal approximation also assumes that one has the maximum likelihood estimate for the parameter and the observed or expected information matrix. Even normal posterior approximations and maximum likelihood calculations can require heavy computation. Newton-Raphson type algorithms for maximum likelihood estimation require several scans of the dataset, at least one for each iteration. When some observations also have missing data, the algorithms (EM, for example) likely will demand even more scans. For some models, dataset sizes, and applications these approximation methods may work and be preferable to a full Bayesian analysis. This will not always be the case and so the need exists for improved techniques to learn accurately from massive datasets.

Summaries of results from a Bayesian data analyses often come in the form of expectations such as the marginal mean, variance, and covariance of the parameters of interest. We compute the expected value of the quantity of interest, $h(\theta)$, using

$$E(h(\theta)|x_1, \cdots, x_n) = \int h(\theta) f(\theta|x_1, \cdots, x_n) d\theta \tag{5.1}$$

where $f(\theta|\mathbf{x})$, is the posterior distribution of the parameters given the observed data. Computation of these expectations requires calculating integrals that, for all but the simplest examples, are difficult to compute in closed form. Monte Carlo integration methods sample from the posterior, $f(\theta|\mathbf{x})$, and appeal to the law of large numbers to estimate the integrals,

$$\lim_{m\to\infty} \frac{1}{m} \sum_{i=1}^{m} h(\theta_i) = \int h(\theta) f(\theta|x_1, \cdots, x_n) d\theta \tag{5.2}$$

where θ_i are m independent and identically distributed (iid) draws from $f(\theta|\mathbf{x})$.

The ability to compute these expectations efficiently, therefore, is equivalent to being able to sample efficiently from $f(\theta|\mathbf{x})$. Sampling schemes are often difficult enough without the burden of large datasets. The additional complexity of massive datasets usually causes each iteration of the Monte Carlo sampler to be slower. When the number of iterations already needs to be large, efficient procedures within each iteration are essential to timely delivery of results.

Example 5.1 (Logistic regression)

In prediction problems in which we wish to predict a binary response, Y, from a set of d predictors, $\mathbf{x}$, linear logistic regression models are often useful. In this modeling framework we assume that Y is a Bernoulli random variable with probability parameter, $p(\mathbf{x})$. That is, given $\mathbf{x}_i$, Y_i will be 1 with probability $p(\mathbf{x}_i)$ and 0 otherwise. The linear logistic assumption assumes that

$$\log \frac{p(\mathbf{x})}{1-p(\mathbf{x})} = \beta_0 + \sum_{j=1}^{d} \beta_j x_j. \tag{5.3}$$

Given a dataset composed of N pairs, $(y_i, \mathbf{x}_i)$, Bayes' theorem produces the form for the posterior distribution of β,

$$f(\boldsymbol{\beta}|\mathbf{x},\mathbf{y}) \propto f(\boldsymbol{\beta}) \prod_{i=1}^{N} \left(1 + e^{-\left(\beta_0 + \sum_{j=1}^{d} \beta_j x_j\right)}\right)^{-y_i} \left(1 + e^{\beta_0 + \sum_{j=1}^{d} \beta_j x_j}\right)^{y_i - 1} \tag{5.4}$$

where $f(\boldsymbol{\beta})$ is the prior distribution for $\boldsymbol{\beta}$, often a multivariate normal density with mean 0 and a diagonal covariance matrix with large variances (*e.g.* 10^4). Algorithms for sampling from this posterior resort to Markov Chain Monte Carlo methods. Each iteration computes $f(\boldsymbol{\beta}|y,\mathbf{x})$ (more often, its full conditional distributions) for a sampled value of $\boldsymbol{\beta}$. However, an evaluation of $f(\boldsymbol{\beta}|y,\mathbf{x})$ or its full conditionals requires a scan of the dataset to produce the linear combination of predictor variables for each observation. MCMC methods already require a large number of iterations. Performance deteriorates further when each iteration scans the full database, potentially reading from a disk those observations unable to fit into the computer's main memory.

Our goal then is to develop a procedure that uses a portion of the dataset to construct an initial posterior distribution. Then with one, maybe two, scans of the remaining dataset, reconstruct the posterior as if we had conditioned on the entire dataset in the first place.

5.2.1 Importance sampling

Importance sampling is a general Monte Carlo method for computing integrals. As previously mentioned, Monte Carlo methods approximate integrals of the form 5.1. Equation 5.2 depends on the ability to sample from $f(\theta|\mathbf{x})$. When a sampling mechanism is not readily available for the "target distribution", $f(\theta|\mathbf{x})$, but one is available for another "sampling distribution", $g(\theta)$, we can use importance sampling. Note that for 5.2 we can write

$$\begin{aligned} \int h(\theta)f(\theta|x_1,\ldots,x_n)d\theta &= \int h(\theta)\frac{f(\theta|\mathbf{x})}{g(\theta)}g(\theta)d\theta \qquad (5.5) \\ &= \lim_{m\to\infty}\sum_{i=1}^{m} h(\theta_i)w_i \qquad (5.6) \end{aligned}$$

where θ_i is a draw from $g(\theta)$ and $w_i = \frac{f(\theta_i|\mathbf{x})}{g(\theta_i)}$. Note that the expected value of w_i under $g(\theta)$ is 1. Therefore, if we are able to compute the importance sampling weights, w_i, only up to a constant of proportionality, we can normalize the weights to compute the integral.

$$\int h(\theta)f(\theta|x_1,\ldots,x_n)d\theta = \lim_{m\to\infty}\frac{\sum_{i=1}^{m} h(\theta_i)w_i}{\sum_{i=1}^{m} w_i} \qquad (5.7)$$

Naturally, in order for the sampling distribution to be useful, drawing from $g(\theta)$ must be easy. We also want our sampling distribution to be such that the limit converges quickly to the value of the integral. The best choice for the sampling distribution is $g(\theta) \propto |h(\theta)f(\theta|\mathbf{x})|$ (Geweke 1989). With this sampling distribution the Monte Carlo variance reduces to zero. However, such a sampling distribution most likely requires that we know the value of the integral in the first place.

If the tails of $g(\theta)$ decay faster than $f(\theta|\mathbf{x})$ the weights will be numerically unstable. If the tails of $g(\theta)$ decay much more slowly than $f(\theta|\mathbf{x})$ we will frequently sample from regions were the weight will be close to zero, wasting computation time. Second to sampling directly from $f(\theta|\mathbf{x})$, we would like a sampling distribution slightly fatter than $f(\theta|\mathbf{x})$.

In the next section I show that when we set the sampling density to be $f(\theta|g_1)$, where g_1 is a manageable subset of the entire dataset, the importance weights for each sampled θ_i require only one sequential scan of the remaining observations.

5.2.2 Importance sampling for inference in massive datasets

Ideally, we would like to sample efficiently and take advantage of all the information available in the dataset. A factorization of the integrand shows that this is possible when the observations, x_i, are iid given the parameters, θ. Let g_1 and g_2 be a partition of the dataset so that every observation is in either g_1 or g_2.

As noted for 5.1 we would like to sample from $f(\theta|x_1, \ldots, x_n) = f(\theta|g_1, g_2)$. Since sampling from $f(\theta|g_1, g_2)$ is difficult due to the size of the dataset, I consider sampling only from $f(\theta|g_1)$ and using importance sampling to adjust. If θ_i, $i = 1, \ldots, m$ are draws from $f(\theta|g_1)$ then we can estimate the posterior expectation 5.1 as

$$\hat{E}(h(\theta)|g_1, g_2) = \frac{\sum_{i=1}^{m} h(\theta_i)w_i}{\sum_{i=1}^{m} w_i} \tag{5.8}$$

where the w_i's are the importance sampling weights

$$w_i = \frac{f(\theta_i|g_1, g_2)}{f(\theta_i|g_1)}. \tag{5.9}$$

Although these weights still involve the computation of $f(\theta_i|g_1, g_2)$, they actually simplify to a computable form.

$$\begin{aligned}
w_i &= \frac{f(\theta_i|g_1, g_2)}{f(\theta_i|g_1)} \\
&= \frac{f(g_1, g_2|\theta_i)f(\theta_i)}{f(g_1, g_2)} \frac{f(g_1)}{f(g_1|\theta_i)f(\theta_i)} \qquad (5.10) \\
&= \frac{f(g_1|\theta_i)f(g_2|\theta_i)f(g_1)}{f(g_1|\theta_i)f(g_1, g_2)} \qquad (5.11) \\
&= \frac{f(g_2|\theta_i)}{f(g_2|g_1)} \\
&\propto f(g_2|\theta_i) \qquad (5.12)
\end{aligned}$$

Line 5.10 follows from applying Bayes' theorem to the numerator and denominator. Equation 5.11 follows from 5.10 since the observations in the dataset partition g_1 are independent from those in g_2. Conveniently, 5.12 is just the likelihood of the observations in g_2 evaluated at the sampled value of θ. The algorithm shown in figure 5.1 summarizes these results.

As an alternative to importance sampling, $f(\theta|g_1)$ could work, analytically at least, as a proposal distribution within a Metropolis independence sampler. However, computing the acceptance probability almost certainly will require a scan of g_2, which we hope to have to do as few times as possible.

So rather than sample from the posterior conditioned on all of the data, g_1 and g_2, which slows the sampling procedure, we need only to sample from the posterior conditioned on g_1. The remaining data, g_2, simply adjusts the sampled parameter values by reweighting. The for loops in step 5 of figure 5.1 are interchangeable. The trick here is to have the inner loop scan through the draws so that the outer loop only needs to scan g_2 once to update the weights. Although theoretically the same computations take place, in practice physically scanning a massive dataset is far more expensive than scanning a parameter list. However, massive models as well as massive datasets exist so that in these cases scanning the dataset would be cheaper

1. Load as much data into memory as possible to form g_1, taking into account space requirements for the Monte Carlo algorithm.
2. Draw m times from $f(\theta|g_1)$ via direct simulation, Monte Carlo, or Markov Chain Monte Carlo.
3. Purge the memory of g_1.
4. Create a vector of length m to store the log-weights and initialize them to 0.
5. Iterate through the remaining observations. For each observation, x_j, update the log-weights on all of the draws from $f(\theta|g_1)$.

```
For $\mathbf{x}_j$ in the partition $g_2$ do
{
    for $i$ in $1, \ldots, m$ do
    {
        $\log w_i \leftarrow \log w_i + \log f(x_j|\theta_i)$
    }
}
```

6. Rescale to compute the weights

$$\begin{aligned} w_i &\leftarrow \exp\left(\log w_i - \max(\log w_i)\right) \\ w_i &\leftarrow \frac{w_i}{\sum_{i=1}^{m} w_i} \end{aligned}$$

Figure 5.1: Importance sampling for massive datasets

than scanning the sampled parameter vectors. I will continue to assume that scanning the dataset is the main impediment to the data analysis.

We certainly can sample from $f(\theta|g_1)$ more efficiently than from $f(\theta|g_1, g_2)$ since simulating from $f(\theta|g_1)$ will require a scan of a much smaller portion of the dataset. For reasons discussed later, the algorithm works best when g_1 is as large as possible. We also assume that, for a given value of θ, the likelihood is readily computable up to a constant, which is almost always the case. When some data are missing, the processing of an observation in g_2 will require integrating out the missing information. Since the algorithm handles each observation case by case, computing the observed likelihood as an importance weight will be much more efficient than if it was embedded and repeatedly computed in a Metropolis-Hastings rejection probability computation. Placing observations with missing values in g_2 greatly reduces the number of times this integration step needs to occur, easing likelihood computations. Aside from missing data problems, in the logistic regression case, where the full conditionals are not in a readily sampled form, the likelihood is simple to compute. I demonstrate the importance sampling algorithm for posterior inference in linear logistic regression models in example 5.2.

Example 5.2 (Logistic regression, continued)

To implement the sampling algorithm for logistic regression we select observations to form partition g_1. Again, this partition ideally contains as many observations as possible so that the MCMC algorithm can produce enough draws in a reasonable amount of time. Using an MCMC algorithm we draw a sample of size m from $f(\boldsymbol{\beta}|g_1)$. Let $\beta^{(1)}, \ldots, \beta^{(m)}$ represent those draws. BUGS (Spiegelhalter, Thomas, Best, and Gilks 1999) can perform this rather simply when g_1 is not too large. For logistic regression the likelihood of an observation is

$$L(\boldsymbol{\beta}|y, \mathbf{x}) = p(\mathbf{x})^y (1 - p(\mathbf{x}))^{1-y} \tag{5.13}$$

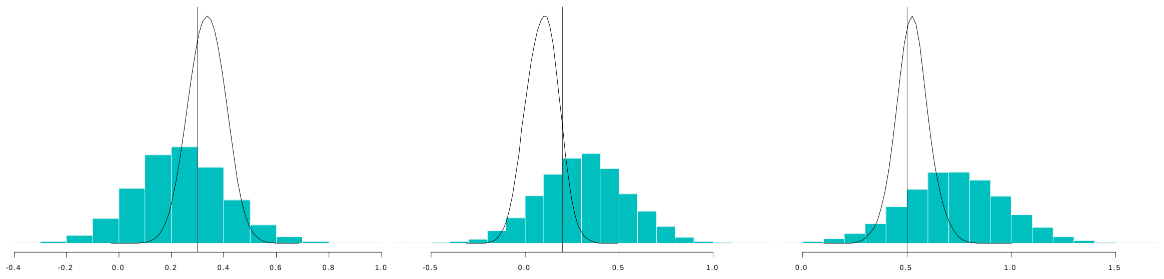

Figure 5.2: Marginal posterior distribution of β_0, β_1, and β_2 before (histogram) and after (smoothed density) adding an additional 10,000 observations. The vertical line indicates the true parameter value.

where $p(\mathbf{x}) = \dfrac{1}{1 + e^{-\left(\beta_0 + \sum_{j=1}^{d} \beta_j x_j\right)}}$.

Initializing the log-weights to zero we can iterate through the observations in g_2 to update the weights associated with each of the sampled $\beta^{(i)}$.

```
Then for $(y, \mathbf{x})_j$ in the partition $g_2$ do
    for $i$ in $1, \ldots, m$ do
        $\eta = \beta_0^{(i)} + \sum_{j=1}^{d} \beta_j^{(i)} x_j$
        $\log w_i \leftarrow \log w_i + y_j \eta - \log(1 + e^{\eta})$
```

After rescaling the weights we have a weighted sample from the posterior distribution conditioned on the entire dataset.

I initially tested this algorithm on a small simulated example. I generated 10,000 observations from a logistic regression model with coefficients $\boldsymbol{\beta} = \{0.3, 0.5, 0.2\}$. I then obtained 10,000 draws from the posterior distribution of $\boldsymbol{\beta}$ using an MCMC algorithm (via BUGS). This sample from the posterior represents the draws from $f(\theta|g_1)$. I then sequentially generated 10,000 additional observations from the logistic regression model while updating the importance sampling weights of the original

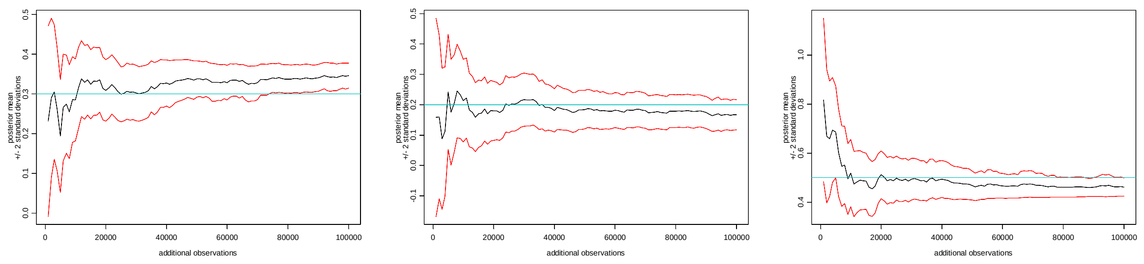

Figure 5.3: Convergence plots of β_0, β_1, and β_2 for the logistic regression example with the addition of 100,000 observations. The horizontal line indicates the true value of the data generating process. The middle curve is the importance weighted mean estimate and the outer bands indicate $\pm$ 2 standard deviations.

draws from the posterior distribution. Figure 5.2 shows histograms of the marginal posterior distributions conditioned on the original 10,000 observations. The smoothed densities are the importance weighted densities incorporating the additional 10,000 observations.

Figure 5.3 shows the behavior of the posterior mean as we incorporate up to 100,000 additional observations. With the first several thousand additional observations we see a large decrease in the variability of the parameter estimates. After adding 20,000 new observations the rate of decrease in variance slows down. From there on we seem unable to extract any more information from the additional observations.

As noted earlier, importance sampling is most efficient when the sampling distribution is close to the distribution from which we wish to sample. Since the sampling distribution conditions on a fairly large portion of the dataset, it likely places mass in similar regions as the target distribution. Since all of the terms are positive in the familiar variance relationship

$$\mathrm{Var}(\theta|g_1) = \mathrm{E}(\mathrm{Var}(\theta|g_1, g_2)) + \mathrm{Var}(\mathrm{E}(\theta|g_1, g_2)), \tag{5.14}$$

the posterior variance with the additional observations in g_2 on average will be smaller than the posterior variance conditioned only on g_1. The addition of g_2 can increase the variance (see Spiegelhalter and Cowell (1992) for an example) but we assume that g_2 is large enough so that we achieve the averaging effect. Therefore, although the location of the sampling distribution should be close to the target distribution, its spread will most likely be wider than that of the target, preventing unstable importance weights.

As more additional observations become available, that is, the size of g_2 grows, the posterior distribution becomes much narrower than the sampling distribution, $f(\theta|g_1)$. The result of this narrowing is that the weights of many of the original draws from the sampling distribution approach 0 and so we have few effective draws from the target density.

Kong, Liu, and Wong (1994) use importance sampling in a similar fashion for missing data problems. In that work they propose a rule of thumb for measuring the effective sample size (ESS) from an importance sampler as

$$\text{ESS} = \frac{m}{1 + \text{Var}(w)} \approx \frac{\left(\sum w_i\right)^2}{\sum w_i^2} \tag{5.15}$$

where $\text{Var}(w)$ indicates the variance of the importance sampling weights, w_i. We can estimate this by substituting a variance estimate $m^2 s^2 \left(\frac{w_i}{\sum_j w_j}\right)$ where $s^2(\cdot)$ is the sample variance of the normalized weights. Figure 5.4 shows the decay of the effective sample size as we add additional observations. After 20,000 the effective sample size reached 52, a huge decrease from $m = 10,000$ from which it started. Figure 5.4 also contains a plot of the number of draws accounting for 99% of the observational weight. After 20,000 additional logistic regression observations, less than 2% of the draws from $f(\boldsymbol{\beta}|\mathbf{y}, \mathbf{x})$ are accounting for 99% of the weight for computing the posterior quantities. The number of initial draws, m, the relative size of g_1 and g_2, and the number of parameters all contribute to whether the effective sample size becomes small enough to be problematic.

The following theorem concerning the variance of the important sampling weights

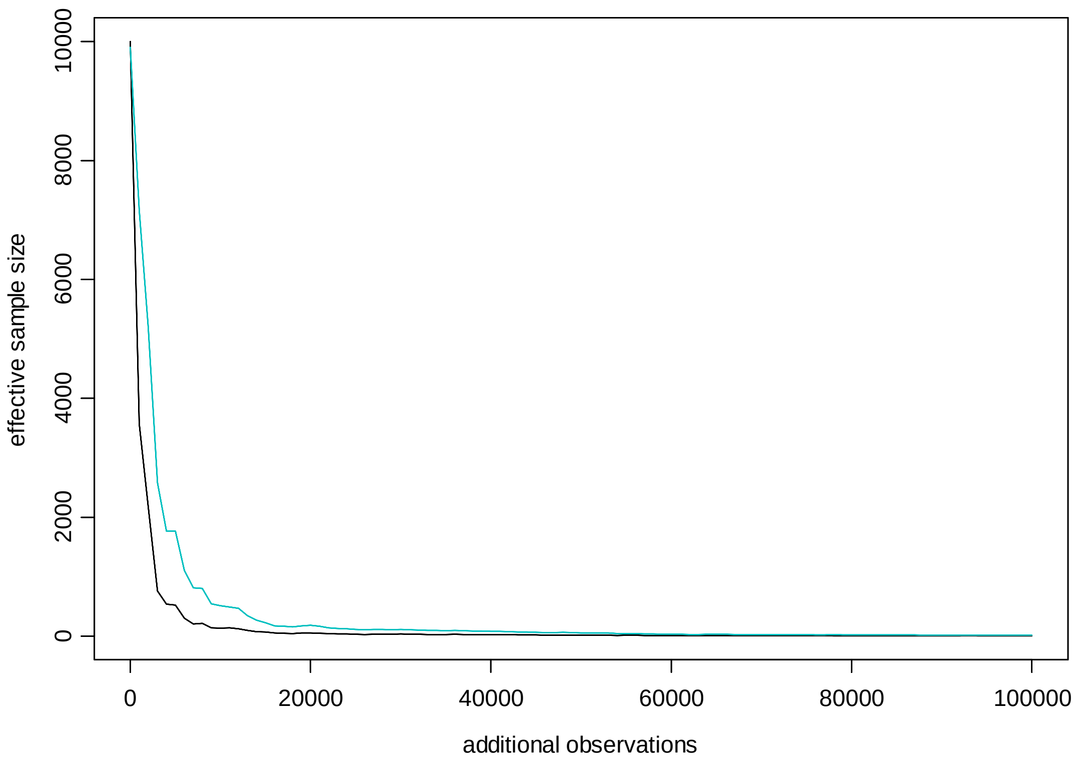

Figure 5.4: The reduction in effective sample size with the addition of 100,000 observations. The lower curve computes ESS and the higher, lighter curve indicates the number of observations accounting for 99% of the observational weight.

can help us gauge the effect of these problems in practice. The theorem assumes that we observe a finite set of multivariate normal data, $\mathbf{y}_i$. As before we will partition the $\mathbf{y}_i$'s into two groups, g_1 and g_2. To make accurate inference for μ we will be concerned about the variance of the importance sampling weights, $\frac{\phi(\mu|g_1,g_2,\Sigma)}{\phi(\mu|g_1,\Sigma)}$, where $\phi(\cdot)$ is the normal density function. The theorem gives the variance of these importance sampling weights averaged over all possible datasets with a flat prior for μ.

Theorem 5.1 *If, for* $i = 1, \ldots, n$*,*

1. $\mathbf{y}_i \sim \mathrm{N}_d(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ *with known covariance* $\boldsymbol{\Sigma}$*,*
2. $g_1 = \{\mathbf{y}_1, \ldots, \mathbf{y}_{n_1}\}$ *and* $g_2 = \{\mathbf{y}_{n_1+1}, \ldots, \mathbf{y}_{n_1+n_2}\}$*, and*
3. $\boldsymbol{\mu} \sim \mathrm{N}_d(\boldsymbol{\mu}_0, \boldsymbol{\Lambda}_0)$

then

$$\lim_{\boldsymbol{\Lambda}_0^{-1} \to \mathbf{0}} \mathrm{E}_{g_2}\mathrm{E}_{g_1}\mathrm{Var}_{\boldsymbol{\mu}|g_1,\boldsymbol{\Sigma}} \left(\frac{\phi(\boldsymbol{\mu}|g_1, g_2, \boldsymbol{\Sigma})}{\phi(\boldsymbol{\mu}|g_1, \boldsymbol{\Sigma})} \right) = \left(\frac{n_1 + n_2}{n_1} \right)^d - 1 \tag{5.16}$$

Proof: See appendix A.

□

Theorem 5.1 basically says that in the multivariate normal case with a flat prior the variance of the importance sampling weights is $\left(\frac{n_1+n_2}{n_1}\right)^d - 1$ on average. These results hold approximately in the non-normal case if the posterior distributions involved are approximately normal, often the case when n_1 and n_2 are reasonably large. As we should expect, when $n_2 = 0$ the variance of the weights is 0. As n_2 increases the variance increases at a rate of $O(n_2^d)$. This is unfortunate in our case since we would like to use this method for large values of n_2 and high dimensional problems. Looking at this result from the effective sample size point of view we see that

$$\mathrm{ESS} \to m \left(\frac{n_1}{n_1 + n_2} \right)^d . \tag{5.17}$$

If we draw m times from the sampling distribution when the size of the second partition g_2 is equal to the size of the first partition g_1, the effective sample size is decreased by a factor of 2^d. Given the largest g_1 that we can handle computationally, if we have an idea of the reduction in ESS that we are willing to accommodate, then we can use theorem 5.1 to decide how to limit the size of g_2. If we are willing to handle a reduction of $p \in (0, 1)$ in our effective sample size so that ESS $= p \times m$ then we can choose n_2 as

$$n_2 = \left(\frac{1}{\sqrt[d]{p}} - 1 \right) n_1. \tag{5.18}$$

In the next section I propose a simple fix to the few effective draws problem. By adapting the importance sampling distribution to focus on regions of the parameter space with non-trivial posterior mass, we can control our effective sample size as n_2 increases.

5.2.3 An improved algorithm using adaptive importance sampling

After drawing a sample of m parameter values from $f(\theta|g_1)$ and scanning g_2 to update the weights we may realize that many of these draws have very small weight. When this happens the effective sample size is small and our Monte Carlo estimates have high variance. Upon realizing the small effective sample size we might choose to generate more draws from $f(\theta|g_1)$ increasing m. Still many of these new draws will have weight close to 0. Unfortunately, in order to determine the importance sampling weight associated with a parameter value we need to scan g_2. When the effective sample size is small we will waste a scan of g_2 computing weights that end up extremely close to 0. However, using the information from previous draws we may be able to guess whether the importance weight will be substantial.

Assume that we have an initial set of draws from $f(\theta|g_1)$ and their associated weights. This initial set of draws should contain a lot of predictive information as to whether a new draw will have weight larger than some threshold. If we can construct a predictor that can quickly determine that a new draw is simply going to have a small weight we can eliminate the waste of computation on draws from the tail of $f(\theta|g_1, g_2)$.

Adaptive importance sampling is the process of learning from previously weighted draws to adjust the sampling density to decrease the Monte Carlo error in the estimates. The VEGAS algorithm (Lepage 1980, Lepage 1978), used extensively in computational physics for 20 years, constructs a sampling density where each dimension of the parameter vector is independent. For example, assume that we wish to compute the integral

$$\int\int\int f(x,y,z)\,dz\,dy\,dx.$$

Further assume that we will use a separable importance sampling distribution of the form $g(x,y,z) = g_x(x)g_y(y)g_z(z)$. VEGAS adapts the components of $g(x,y,z)$ to bring the importance sampling distribution closer to proportionality with $|f|$. The optimal separable sampling distribution is

$$g_x(x) \propto \left[\int\int \frac{f^2(x,y,z)}{g_y(y)g_z(z)}\,dz\,dy\right]^{\frac{1}{2}}. \tag{5.19}$$

and similarly for $g_y(y)$ and $g_z(z)$. Equation 5.19 suggests how to adapt the sampling density to be more efficient. We can estimate the components of $g(x,y,z)$ by subdividing each dimension into K bins and use 5.19 to update the mass in each of those bins. VEGAS rapidly concentrates weight on the important dimensions and regions. Clearly this algorithm will perform best when the optimal sampling distribution is separable and performance deteriorates as the sampling distribution departs from separability. Although this algorithm is popular in elementary particle physics it seems to have received little attention in Bayesian analysis.

Other researchers have considered adaptive importance sampling (Oh and Berger 1992, West 1992, West 1993, Givens and Raftery 1996). They offer algorithms that sample from a parametric or non-parametric density that on occasion adapts to the draws obtained so far. However, they all confess to work only in low dimensions, less than 10, and are not concerned with a large dataset in addition.

After one round of sampling from $f(\theta|g_1)$ and one pass through the rest of the dataset, we have m pairs $(\theta, w)_i$ where the w_i are normalized to sum to 1. To minimize the amount of wasted computation, we would like to build a simple predictive model to identify new draws that are likely to have weight near zero. That is, we can construct a classifier that predicts the indicator $I(w > t)$, where t is a threshold parameter, with the components of θ as predictors. In this problem the misclassification of a draw with a large weight is costly. Our main motivation is to trim many of the draws with negligible weight. I propose a tree-based and direct trimming method and argue that the direct trimming classifiers might have the advantage.

Given the classification task at hand, initially we might believe that constructing a classification tree would be ideal. If the posterior is complex, possibly banana shaped in a high-dimensional space, the tree might still be able to capture the highly weighted regions. Since we are competing against a penalty that is growing at a rate of $O(n^d)$ with the addition of observations, we need an adaptation that is learning the regions of negligible weight at the same rate. However, to achieve this goal the tree might not even need to be very deep. In fact, the best classification tree with only two terminal nodes, that is a split on only one dimension of θ, is likely to eliminate many of the negligible draws. In fact we could fit a one-split decision tree to each of the d axes. Each axis' tree would determine a region that it predicted to have little posterior mass. Figure 5.5 demonstrates this idea, the shaded regions being the rejection regions. A new draw could not be in any of these regions in order to be admitted to the Monte Carlo sample. This type of marginal computation is in the same style as the VEGAS algorithm. Both perform computations on the margin to conserve computation and storage and assume that working on the margin alone yields an adequate sampling density.

For well-behaved (unimodal, near elliptical) posterior distributions, building a tree

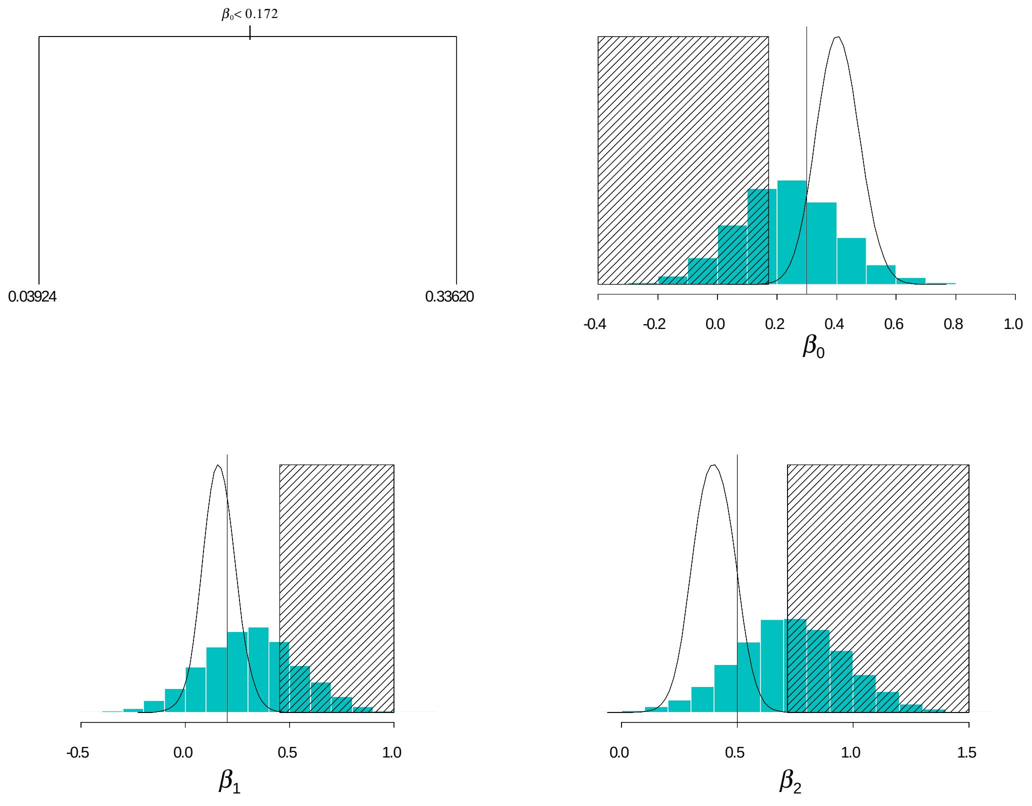

Figure 5.5: An example axis classifier and the rejection regions on each parameter axis

on each axis might be sufficient to overcome the $O(n^d)$ rate. Although the tree method seems attractive since it is easily computable and automated, it may be overkill for our designs. Consider the following proposed algorithm for developing a rejection region. First, I sort the Monte Carlo draws on the j^{th} dimension of $\boldsymbol{\theta}$. Then I record the α and $1 - \alpha$ weighted quantiles of $\theta^{(j)}$, where the weights are the importance sampling weights, and trim values outside of these bounds. If, for example, I let $\alpha = 0.005$ then I have only trimmed 1% of the observational weight but, according to figure 5.4, after 20,000 iteration this could mean freeing the memory of 98% of the Monte Carlo draws. Repeating this process on the other parameter dimensions could realize polynomial savings to compete with the $O(n^d)$ importance sampling penalty in equation 5.17.

Note that the tail trimming classifier does assume that the posterior is unimodal to make the performance gains, a reasonable assumption for most models conditioned on massive datasets. Furthermore, the classifier forms decision boundaries perpendicular to the axes. For strongly correlated data this is not ideal. When the data are strongly correlated, one could perform the quantile trimming on the principal component axes of the data distribution. However, this clearly requires more computation both in the trimming and later in assessing whether to reject a Monte Carlo draw. This method would be inefficient and fairly useless in the case of a strongly banana shaped posterior in high dimensions. Such distributions with a lot of curvature can take up little mass inside the hypercube formed by the quantile limits. For the most common situations, quantile trimming on the axes is likely to be accurate and is always fast ($O(dn \log n)$ although a complete sort is not necessary). Competing against this kind of computational efficiency would be difficult.

From here on I will assume that the adaptation of the importance sampler incorporates the weighted α-quantiles from each dimension of $\boldsymbol{\theta}$. Let $I(\theta)$ indicate whether a parameter value falls inside (1) or outside (0) the weighted α-quantile bounds on each axis. We can generate a new set of candidate draws and use $I(\theta)$ to eliminate those with expected negligible weight. This will save us from having to use $f(g_2|\theta)$ to determine whether a draw will have substantial weight. However, we may still be wasting expensive Monte Carlo iterations producing draws in the tails of $f(\theta|g_1, g_2)$. Rather than use a draw-check-draw-check method, we could modify our Monte Carlo sampling scheme to draw instead from $f_\alpha(\theta|g_1)$, a mixture of $f(\theta|g_1)$ and a truncated version of it where α here is the mixing proportion.

$$f_\alpha(\theta|g_1) = \alpha \frac{f(\theta|g_1)I(\theta)}{\int f(\theta|g_1)I(\theta)d\theta} + (1-\alpha)f(\theta|g_1) \tag{5.20}$$

Setting α to be large to favor the truncated version will concentrate draws in the region that might offer more substantial importance weights. For a Metropolis-Hastings Monte Carlo algorithm, sampling from the truncated portion of the mixture simply alters the proposal mechanism.

Recomputing the importance weights for *theta*'s drawn from the mixture sampling distribution in 5.20 yields

$$w_i = \frac{f(g_2|\theta_i)}{\alpha I(\theta_i) + (1-\alpha)\int f(\theta|g_1)I(\theta)d\theta} \tag{5.21}$$

The integral in the above expression is the probability of generating a draw from $f(\theta|g_1)$ that the weight trimming predictor, $I(\theta)$, accepts. We can use the original unweighted draws from $f(\theta|g_1)$ as an estimate. If this probability is small then $I(\theta)$ selected a region that $f(\theta|g_1)$ does not frequently sample. Also note that if $\alpha = 1$ then we will always sample from the truncated $f(\theta|g_1)$, $I(\theta)$ will always be 1, and the w_i will remain as before. Although setting the mixing proportion $\alpha = 1$ does not constitute a precise sampling scheme, if the quantile trims are small, as the data accumulates the difference should be inconsequential.

These results suggest the adaptive sampling algorithm shown in figure 5.6. After an initial set of m draws θ_j from a posterior distribution conditioned on a small dataset, $f(\theta|g_1)$, the algorithm will continue adding observations from g_2 until the

Partition the dataset of N observations x_i into g_1 and g_2. Obtain an initial set of m draws θ_j from $f(\theta|g_1)$ with associated log-weights $\log w_j = 0$. Initialize $i = 1$, α as the trimming factor, and δ as the minimum acceptable effective sample size fraction.

```
while $i \leq N$ do
    while ESSf $> \delta$ and $i \leq N$ do
        Select $x_i$ from $g_2$
        For $j = 1, \ldots, m$ do
            $\log w_j \leftarrow \log w_j + f(x_i|\theta_j)$
        $i \leftarrow i + 1$
    Compute the weighted $\alpha$-quantiles on each dimension of $\theta$.
    Discard $\theta_j$ outside the quantile bounds.
    Sample replacement $\theta_j$ from $f(\theta|g_1)$ truncated at the quantiles.
    Reweight the new draws for $k$ in $1, \ldots, i$ do
        Select $x_k$ from $g_2$
        For $j \in$ new draws do
            $\log w_j \leftarrow \log w_j + f(x_k|\theta_j)$
Return the $\theta_j$'s weighted by $w_j$.
```

Figure 5.6: Adaptive importance sampling for inference in massive datasets

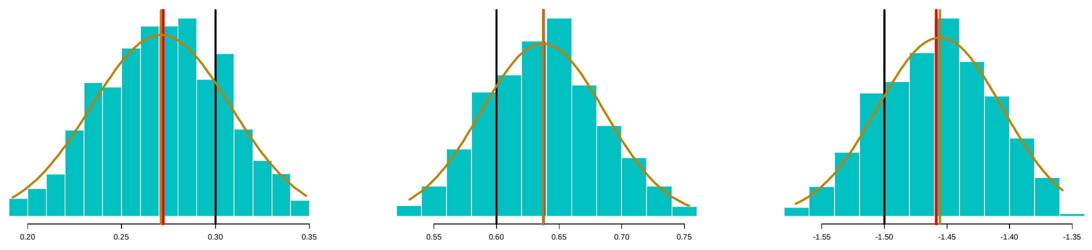

Figure 5.7: Adaptive importance sampling estimates of β_0, β_1, and β_2 from the logistic regression example. The histogram is the importance weighted estimate of $f(\boldsymbol{\beta}|\mathbf{x})$. The dark vertical line outside the middle of the histogram indicates the true parameter value. The vertical line near the center of the histogram marks the posterior expectation. The superimposed density plot is the mle normal approximation to the posterior, its mean partially obscured by the posterior expectation.

effective sample size fraction 5.22 drops below a certain threshold, δ.

$$\text{ESSf} = \frac{\left(\sum w_j\right)^2}{m \sum w_j^2} \tag{5.22}$$

At that point we can extract the weighted marginal α-quantiles from each axis. After rejecting those θ_j which are outside of these bounds, we can replace them with draws from the importance sampling distribution truncated at the quantiles. The new draws need reweighting using the subset of g_2 already examined. This does require a scan of those observations previously scanned. Again we continue until the ESS fraction drops too low. The algorithm proceeds until all the observations in g_2 are incorporated while preserving the lower bound on the ESS fraction. At the end we will have a set of weighted draws from the posterior $f(\theta|g_1, g_2)$ (approximately due to the presumably negligible truncation when $\alpha = 1$).

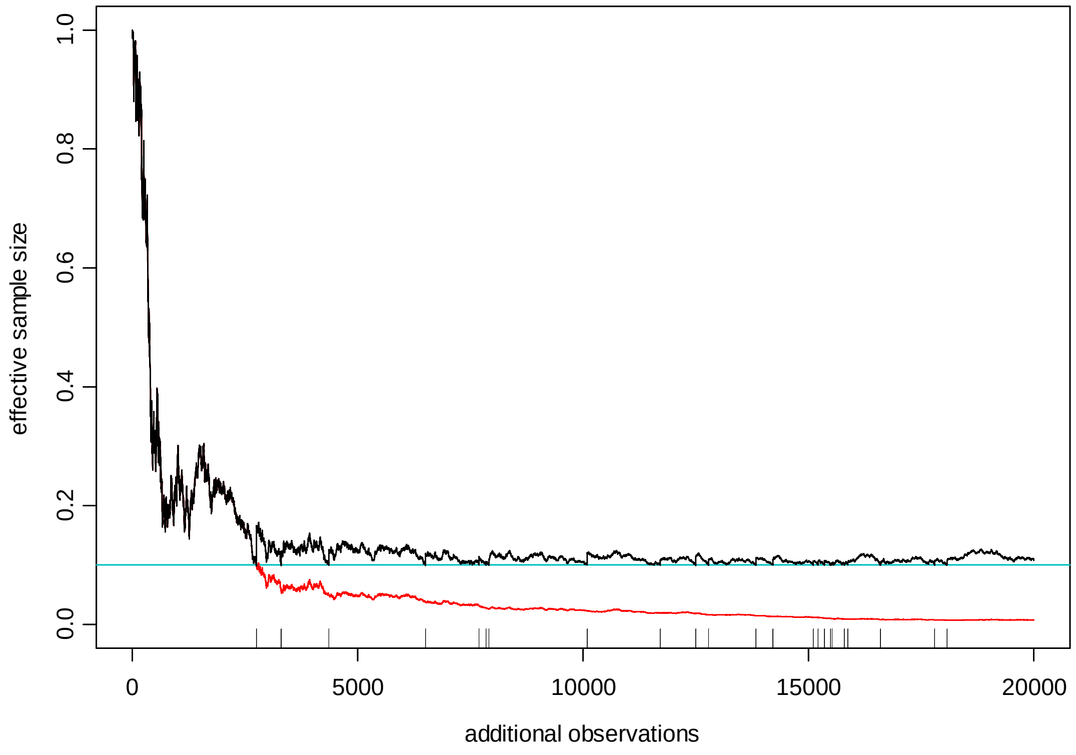

Figure 5.8: Effective sample size as a function of the additional observations during adaptive importance sampling. The upper curve indicates the ESS with each additional observation, maintained above the 10% threshold. The lower curve is the ESS without adapting the importance sampler. Adaptation points are marked by the rug plot.

5.2.4 Evaluation and modifications to the adaptive sampling algorithm

I tested the adaptive importance sampling algorithm on the logistic regression example. I generated 21,000 observations according to

$$\begin{aligned}
&x_1, x_2 \sim U(0,1) \\
&y \sim \text{Bern}(\text{logit}^{-1}(0.3 + 0.6x_2 - 1.5x_2)).
\end{aligned}$$

Although I want to draw from the posterior conditioned on all 21,000 observations, I used a Metropolis-Hastings algorithm to sample from the posterior conditioned on only the first 1,000 observations. The adaptive importance sampling procedure incorporated the remaining 20,000 observations. I set the lower bound on the effective sample size fraction at 0.10 and always maintained 10,000 draws from the importance sampling distribution. Figure 5.7 shows a histogram of the marginal posterior distribution of the three regression coefficients using adaptive importance sampling. The overlaid density is the normal approximation to the marginal posterior conditioned on all 21,000 observations. These plots indicate that we can perform MCMC steps, drawing from the posterior conditioned on a small amount of data, and incorporate 20 times that amount using only a few partial scans of the dataset. Rescanning is not completely inexpensive. In this example they are equivalent to scanning the dataset 14 times.

The effective sample size is still of interest. Figure 5.8 shows the trend of the effective sample size fraction as more data enter the posterior. Every time ESS drops below 10% the algorithm adapts the importance sampler by truncating the conservative $\alpha = 0.001$ weighted quantile from each axis. The rug plot marks the adaptation points, which occurs with every 870 additional observations on average. Without the adaptation steps, ESS continues to drop toward zero as the lower curve indicates. The adaptation, however, removes observations with little weight. In this example each adaptation eliminated between 5% and 15% of the Monte Carlo draws. Their removal stabilized the variance of the importance sampling weights and maintained an adequate ESS. Although I obtain 10,000 draws from the posterior conditioned on 1,000 observations, at the end of the process I have what amounts to 1,000 draws from the posterior conditioned on all 21,000 observations.

There are two ways in which we might want to optimize the algorithm. First, although it is easy to maintain ESS at about 10%, at this level the importance sampler is mildly inefficient. Second, the adaptation is not entirely cheap and so we would like to reduce the frequency of adaptations. These two are competing goals. I found that increasing the lower bound on ESS increased the frequency of adaptation, sometimes after adding each new observation. However, we not only have control over ESS but we can also alter the trimming step by tuning the quantile parameter α and the trimming process. The fear in setting an aggressive α is that the algorithm might trim away the regions with high posterior mass. It may be possible to learn when and where we can be more aggressive.

Notice that in both the recent results shown in figure 5.7 and the illustration of the truncation sampler in figure 5.5, wasted draws from the importance sampling distribution tend to be concentrated in one tail. This implies that trimming different quantiles from each tail might be more sensible. Consider the following modification to the adaptive importance sampling algorithm. After the ESS fraction becomes too small we will compute four quantiles, two “weak” quantiles and two “strong” quantiles. On each margin we will choose one side to strongly truncate by removing α_s weight and on the other side we will weakly truncate by removing α_w weight. I will use the heuristic of strongly truncating the tail from which trimming α_s weight would remove the largest number of Monte Carlo draws. That is, I will compute the weighted α_s-quantile and the $(1-\alpha_s)$-quantile. If the number of draws less than the α_s-quantile is larger than the number of draws greater than the $(1-\alpha_s)$-quantile I will truncate the left tail at the α_s-quantile and the right tail at the $(1-\alpha_w)$-quantile. Figure 5.5 demonstrates this principle. The shaded boxes cover the regions where the

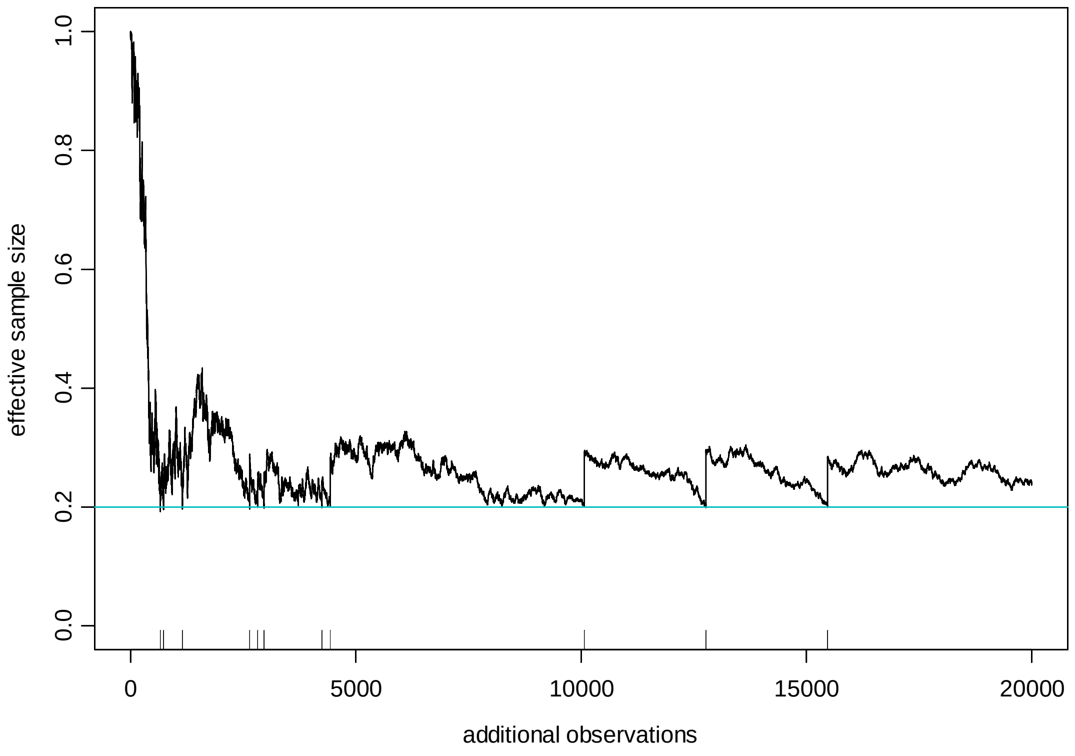

Figure 5.9: Effective sample size as a function of the additional observations during aggressive adaptive importance sampling. It maintains a higher ESS fraction of 20% with fewer adaptations.

importance sampling weights are small but which contain a lot of Monte Carlo draws.

I repeated the logistic regression experiment setting $\alpha_s = 0.025$, $\alpha_w = \frac{1}{m}$ (the minimum), and the lower bound on the ESS fraction at $\delta = 0.20$. By aggressively trimming only one side and not trimming the other we might be able to achieve increase effective sample size and less frequent adaptations. As opposed to every 870 on the previous example, on average this one adapts with every 1,800 additional observations. Figure 5.9 shows the effective sample size fraction trend. As hoped for, with these settings ESS stays much higher with fewer adaptations. This generates

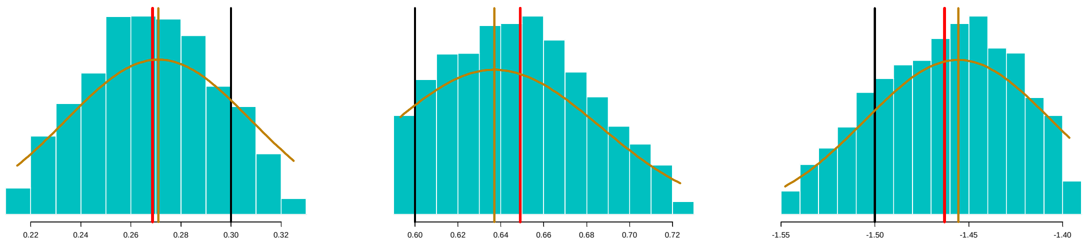

Figure 5.10: Aggressive adaptive importance sampling estimates of β_0, β_1, and β_2 from the logistic regression example. The non-central vertical black lines indicate the true parameter values. The density plot is the mle normal approximation.

a more efficient importance sampler with less computation. This example needed the equivalent of 3 scans of the dataset compared with 14 scans from the previous example.

However, the aggressive trimming is not without dangers. Figure 5.10 shows the marginal posterior distributions. The effect of the truncation is much more apparent. Of particular concern is the center picture, the marginal distribution for β_1. We see that the truncation point has come close to the true value of 0.6. With the data that it observed so far, at least one of the adaptations chose to aggressively trim the left tail. This may be unavoidable under this truncation selection scheme. Solutions to this problem include

1. decreasing the strong truncation quantile,

2. increasing the number of Monte Carlo draws for more accurate quantile estimation,

3. decreasing the ESS fraction bound to add more observations between adaptations and thereby decreasing the variability of the weight of a particular draw,

4. periodically readjusting the bounds allowing the acceptance region to expand (e.g. estimate the marginal as a truncated normal and readjust the bounds to $\mu \pm 3\sigma$),

5. replace the quantile truncation with $\hat{\mu} \pm 4\hat{\sigma}$ where $\hat{\mu}$ and $\hat{\sigma}$ are marginal importance weighted estimates of the mean and standard deviation, and

6. sampling from the mixture in 5.20 with a mixing parameter less than 1.

5.3 Likelihood clustering

In this section I present some preliminary ideas and results on a new method for performing inference in a massive dataset. When faced with a massive dataset the likely response from a statistician is to select a random sample and fit a model to the sample. The possibility exists, however, that with a little additional effort one could compose a more principled, stratified subsample that outperforms simple random sampling but avoids iterating many times over the full dataset to fit the desired model.

The principle that drives this work is that many observations in a massive dataset might be redundant, either copies or near copies of other observations. DuMouchel et al. (1999) proposed a method that builds a pseudo-dataset that matches the moments of the original dataset. They suggest that if the likelihood is smooth then "data squashing" will reduce the redundancy while preserving much of the information contained in the full dataset. Their method is independent of the model under consideration, but shows only a moderate improvement over simple random sampling.

With a likelihood-based model in mind we may be able to stratify the observations into clusters with similar likelihood, improving upon the data squashing approach as well as the simple random sample. More precisely, if many observations have the same value of the likelihood function for all values of the parameters, we would like to cluster them and carry out a likelihood based inference using a weighted likelihood of the clustered data. As in the previous sections I try to avoid the expensive scans of the dataset. The process of "likelihood clustering" reduces redundancy in the massive dataset producing a smaller weighted dataset for which standard statistical tools are tractable.

5.3.1 Motivation for likelihood clustering

Both Bayesian and likelihood inference rely on computations involving the familiar likelihood function.

$$L(\theta|\mathbf{x}) = \prod_{i=1}^{N} f(x_i|\theta) \tag{5.23}$$

For the massive dataset problem this is difficult to compute for one value of θ and even more difficult to maximize. Observations in the massive dataset might have partially redundant information. Two observations, x_i and x_j, have redundant information if for all values of θ, $f(x_i|\theta) = f(x_j|\theta)$. This is surely the case if $x_i = x_j$. However, even if the two values are not equal they may still have the same or approximately the same value for the likelihood at all values of θ. Furthermore, we are rarely interested in *all* values of θ but rather just the region for which the likelihood is large or the posterior mass is high.

The goal at present then is to partition or cluster the data $\mathbf{x}$ into smaller sub-datasets for which the likelihood function evaluated at likely values of θ has approximately the same value. I will partition the dataset into K subsets, $g_1, \ldots, g_K$ so that for all x_i in the cluster g_k we have

$$L(\theta|g_k) = \prod_{x_i \in g_k} f(x_i|\theta) \approx f(x^{(k)}|\theta)^{n_k} \tag{5.24}$$

where n_k is the number of observations in g_k and $x^{(k)}$ is a pseudo-datapoint representing the k^{th} cluster. From a Bayesian perspective we might want to select the pseudo-datapoints $x^{(k)}$ so that

$$\{x^{(1)}, \ldots, x^{(K)}\} = \arg\min \int \mathcal{L}\left(\prod_{i=1}^{N} f(x_i|\theta), \prod_{k=1}^{K} f(x^{(k)}|\theta)^{n_k}\right) f(\theta|\mathbf{x}) d\theta \tag{5.25}$$

where $\mathcal{L}(\cdot,\cdot)$ is some form of loss function. This says that on average under the posterior the likelihood of the entire dataset is close to the weighted likelihood of the pseudo-dataset. In the experiments in this section I used squared error loss for $\mathcal{L}(\cdot,\cdot)$.

I propose the following method to approximate the solution to 5.25. Suppose that we can get a rough guess for $f(\theta|\mathbf{x})$, say, by using the normal-mle approximation where the mle conditions on a small dataset. For each observation we can then compute the likelihood of the observation at a handful of values for the parameter θ selected randomly or deterministically that seem likely under our rough approximation to the posterior. Then we can call upon standard efficient clustering algorithms, such as the k-means algorithm, to group together observations for which their likelihoods at these points are similar. This will form the K partitions $g_1, \ldots, g_K$. Lastly we can select a representative point or compose a pseudo-datapoint from the observations in the cluster.

Figure 5.11 summarizes these ideas, outlining the components of the likelihood clustering algorithm.

5.3.2 A likelihood clustering algorithm for simple linear regression

In this section I will present an algorithm for likelihood clustering for a simple linear regression model and evaluate it on simulated data.

For simple linear regression we assume that

$$y_i \sim N(\beta_0 + \beta_1 x_i, \sigma^2). \tag{5.26}$$

Least squares algorithms for this model can collect all the sufficient statistics for $\hat{\boldsymbol{\beta}}$ in one pass through the dataset. However, we will be interested in seeing whether estimates from a simple random sample or estimates from a likelihood clustered sample come closest to the least squares estimate.

Let $f(x_i|\theta)$ be the likelihood of an observation, N be the total sample size, and $n \ll N$ be the largest sample size for which we can perform inference on θ.

A likelihood clustering algorithm is then

1. Randomly sample n observations from the full dataset.
2. Select d values of θ that are likely given the n observations.
3. For each observation compute the likelihood at each of the d θ's.
4. Partition the observations in n clusters according to the value of the likelihood at the d θ's. This might be computable in conjunction with the previous step.
5. Create a pseudo-datapoint from each cluster to represent its members.
6. Use standard statistical techniques to estimate θ from the n pseudo-datapoints where each receives weight equal to the number of observations in the respective cluster.

Figure 5.11: The likelihood clustering algorithm

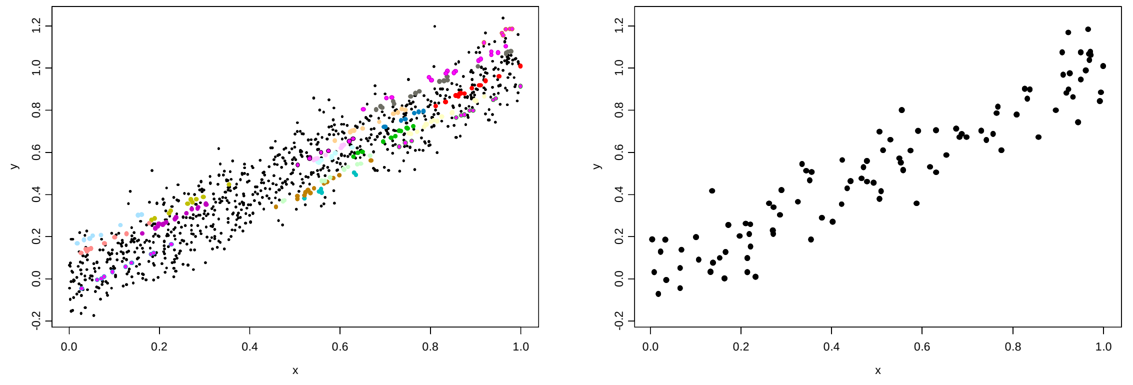

Figure 5.12: Likelihood clusters for the simple linear regression example. The left plot shades a few of the likelihood clusters. Sampling one observation from each cluster produces the plot on the right.

From the principles proposed in the previous section I can use the following steps to construct a pseudo-dataset that could outperform simple random sampling. If the full data set contains N observations then we can get a rough estimate of the posterior mode by fitting a least squares line through a random subsample of size n of the full dataset. This produces a point estimate and standard errors for $\hat{\boldsymbol{\beta}}$ that we can use to approximate $f(\boldsymbol{\beta}|\mathbf{x})$ or at least get an idea where the most likely regression coefficients lie. Evaluating the likelihood for each observation at a small number of these likely parameter values creates a vector of features for each observations. We can then use the k-means algorithm to cluster the observations according to these features. I will select one pseudo-datapoint by randomly selecting one observation from each cluster. A weighted least squares fit to these pseudo-points with weights equal to the size of their cluster should be more competitive than simple random sampling.

To test this I simulated 1,000 observations from a simple linear regression model with $\boldsymbol{\beta} = \{0, 1\}$. I used 100 of them to get an estimate of $\boldsymbol{\beta}$ and the standard errors. For each of the 1,000 observations I computed the likelihood at five parameter values, $\hat{\boldsymbol{\beta}}$ and $\hat{\boldsymbol{\beta}} \pm 3\hat{\text{se}}(\hat{\boldsymbol{\beta}})$. These values are roughly the posterior mode and four values

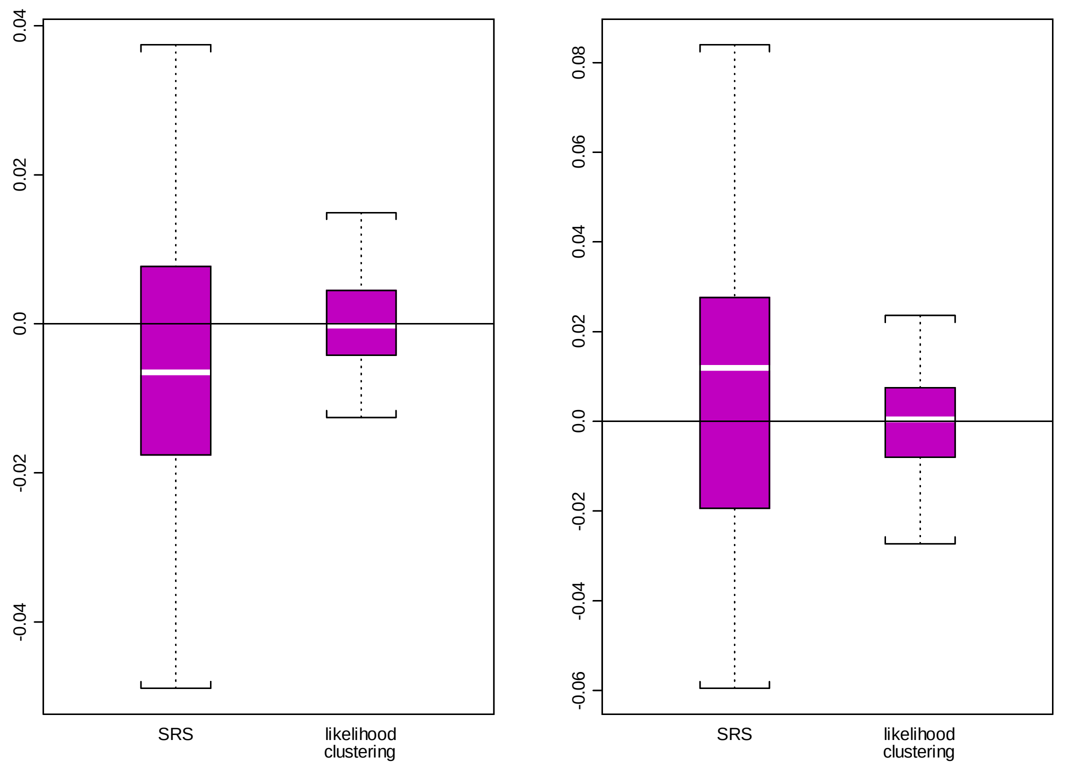

Figure 5.13: Estimation performance of simple random sampling and likelihood clustering. The boxplots show the results over 100 simulations of the difference between the mle using the full dataset and the mle using either simple random sampling or likelihood clustering. The left plot is for β_0 and the right for β_1.

that are three standard deviations away. I then used the k-means algorithm to find 100 clusters and I randomly picked one observation from each cluster as a pseudo-datapoint. The left plot in figure 5.12 highlights a few of those clusters. The selection shown is intuitive. Likelihood clustering tended to group together observations that were linear in a probable direction of β_1. The data squashing would have grouped together observations that were close in terms of x and y but would miss our linear modeling goal. The right picture in figure 5.12 shows the points in the pseudo-dataset. I fit a regression model to the 100 pseudo-datapoints each weighted according to the size of their cluster. I also fit a regression model to a simple random sample of size 100.

Figure 5.13 shows the experimental results for estimating β_0 (left) and β_1 (right) over 100 repetitions. The boxplots show the difference between the estimates from a simple random sample and the likelihood clustering. Estimates close to the zero line are best. Estimates based on simple random sampling had variance four times larger than those using likelihood clustering. In this example likelihood clustering improved accuracy of the estimates by 70% in terms of absolute deviation.

5.4 Discussion

In this chapter I presented some new ideas to create a generation of statistical procedures for use in massive datasets. The first idea is to use a distribution conditioned on a small portion of the dataset as an importance sampling density. Conveniently, the importance weights reduce to the likelihood of the remaining observations, a computation that is usually easy to compute. I also proposed methods specific to this problem for adapting the importance sampling distribution to control the effective sample size. I show that I can effectively sample from a posterior distribution conditioned on a dataset 20 times larger than the one used for the base MCMC algorithm.

The second idea clusters data points in the massive dataset according to their likelihood evaluated at probable values of the parameter. Weighted maximum likelihood on a set of pseudo-datapoints generated from the clusters provides estimates that outperform those from a simple random sample of the dataset. These results show great promise for a general purpose procedure for compressing the dataset into a form that standard fitting procedures can use. If analysts desire more precision this approximation may provide an ideal starting point for the adaptive sampling procedure also introduced in this section.

Further research will investigate optimal and less heuristic clustering and pseudo-datapoint selection methods.

BIBLIOGRAPHY

Abdon, P., A. Lindstrand, and K. Thorngren (1990). Statistical evaluation of the diagnostic criteria for meniscal tears. *International Orthopaedics 14*(4), 341–345.

Aitkin, M. and D. Clayton (1980). The fitting of exponential, Weibull, and extreme value distributions to complex censored survival data using GLIM. *Applied Statistics 29*, 156–163.

Anderson, A. and A. Lipscomb (1986, July-August). Clinical diagnosis of meniscal tears. Description of a new manipulative test. *American Journal of Sports Medicine 14*(4), 291–293.

Barry, O., H. Smith, F. McManus, and P. MacAuley (1983, April). Clinical assessment of suspected meniscal tears. *Irish Journal of Medical Science 152*(4), 149–151.

Bates, J. and C. Granger (1969). The combination of forecasts. *Operations Research Aquarterly 20*, 451–468.

Bauer, E. and R. Kohavi (1999). An empirical comparison of voting classification algorithms: bagging, boosting, and variants. *Machine Learning 36*(1/2), 105–139.

Becker, B., R. Kohavi, and D. Sommerfield (1997). Visualizing the simple Bayesian classifier. In *KDD 1997 Workshop on Issues in the Integration of Data Mining and Data Visualization.*

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition.* Clarendon Press.

Breiman, L. (1996a). Bagging predictors. *Machine Learning 26*, 123–140.

Breiman, L. (1996b). Heuristics of instability in model selection. *Annals of Statistics 24*, 2350–2383.

Breiman, L. (1996c, November). Out-of-bag estimation. Technical report, University of California, Berkeley, Statistics Department.

Breiman, L. (1996d, November). Pasting bites together for prediction in large datasets and online. Technical report, University of California, Berkeley, Statistics Department.

Breiman, L. (1997, December). Prediction games and arcing algorithms. Technical Report 504, University of California, Berkeley, Statistics Department.

Breiman, L. (1998). Arcing classifiers. *The Annals of Statistics 26*(3), 801–849.

Breiman, L. (1999, February). Using adaptive bagging to debias regressions. Technical Report 547, University of California, Berkeley, Statistics Department.

Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone (1984). *Classification and Regression Trees*. Wadsworth.

Breslow, N. (1972). Discussion of "Regression models and life tables" by D.R. Cox (1972). *Journal of the Royal Statistical Society (Series B) 34*, 187–203.

Brutlag, J. and T. Richardson (1999). A block sampling approach to distinct value estimation. Technical Report 355, University of Washington.

Chipman, H., E. George, and R. McCulloch (1998). Bayesian CART model search (with discussion). *Journal of the American Statistical Association 93*, 935–960.

Copas, J. (1983a). Plotting p against x. *Applied Statistics 32*, 25–31.

Copas, J. (1983b). Regression, prediction and shrinkage. *Journal of the Royal Statistical Society, Series B 45*(3), 311–354.

Cox, D. (1972). Regression models and life tables. *Journal of the Royal Statistical Society (Series B) 34*, 187–203.

Cox, D. (1975). Partial likelihood. *Biometrika 62*, 269–276.

Curtin, W., D. O'Farrell, F. McGoldrick, M. Dolan, G. Mullan, and M. Walsh (1992, May). The correlation between clinical diagnosis of knee pathology and findings at arthroscopy. *Irish Journal of Medical Science 161*(5), 135–136.

DeGroot, M. (1970). *Optimal Statistical Decisions*. New York: McGraw-Hill.

Denison, D., B. Mallick, and A. Smith (1996). Bayesian CART. Technical report, Department of Mathematics, Imperial College.

Dickson, E., P. Grambsch, T. Fleming, L. Fisher, and A. Langworthy (1989). Prognosis in primary biliary cirrhosis: model for decision-making. *Hepatology 10*, 1–7.

Dietterich, T. G. (1998). An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. Technical report, Department of Computer Science, Oregon State University.

Dollard, J. and C. Friedman (1979). *Product Integration with Applications to Differential Equations*. Addison-Wesley Publishing Company.

Domingos, P. and M. Pazzani (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning 29*, 103–130.

Drucker, H. (1997). Improving regressors using boosting techniques. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 107–115.

DuMouchel, W., C. Volinsky, T. Johnson, C. Cortes, and D. Pregibon (1999, March). Squashing flat files flatter. Technical report, AT&T Labs-Research.

Efron, B. and R. Tibshirani (1993). *An Introduction to the Bootstrap*. Chapman & Hall.

Elder, J. and D. Pregibon (1996). A statistical perspective on knowledge discovery in databases. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (Eds.), *Advances in Knowledge Discovery and Data Mining*, Chapter 4. AAAI/MIT Press.

Elkan, C. (1997, September). Boosting and naïve Bayes learning. Technical Report CS97-557, University of California, San Diego.

Finney, D. (1971). *Probit Analysis.* Cambridge University Press.

Fisher, R. (1935). The case of zero survivors (appendix to C.I. Bliss 1935). *Annals of Applied Biology 22*, 164–165.

Fleming, T. and D. Harrington (1991). *Counting processes and survival analysis.* New York: Wiley.

Frank, E., L. Trigg, G. Holmes, and I. Witten (1998, April). Naïve Bayes for regression. Technical Report Working Paper 98/15, Department of Computer Science, University of Waikato.

Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Information and Computation 121*(2), 256–285.

Freund, Y. and R. Schapire (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences 55*(1), 119–139.

Friedman, J. (1991). Multivariate adaptive regression splines (with discussion). *Annals of Statistics 19*(1), 1–82.

Friedman, J. (1999a, February). Greedy function approximation: A gradient boosting machine. Technical report, Stanford University, Statistics Department.

Friedman, J. (1999b, March). Stochastic gradient boosting. Technical report, Stanford University, Statistics Department.

Friedman, J., E. Grosse, and W. Stuetzle (1983). Multidimensional additive spline approximation. *SIAM Journal on Scientific and Statistical Computing 4*, 291.

Friedman, J. and P. Hall (1999, May). On bagging and nonlinear estimation. Technical report, Department of Statistics, Stanford University.

Friedman, J., T. Hastie, and R. Tibshirani (1998, August). Additive logistic regression: a statistical view of boosting. Technical report, Stanford University, Statistics Department.

Friedman, J. and W. Stuetzle (1981). Projection pursuit regression. *Journal of the American Statistical Association 76*, 817–823.

Friedman, N., D. Geiger, and M. Goldszmidt (1997). Bayesian network classifiers. *Machine Learning 29*, 131–163.

Friedman, N. and M. Goldszmidt (1996). Building classifiers using Bayesian networks. In *Proceedings of the National Conference on Artificial Intelligence*, Menlo Park, CA, pp. 1277–1284. AAAI Press.

Gelman, A., J. Carlin, H. Stern, and D. Rubin (1995). *Bayesian Data Analysis.* New York: Chapman Hall.

George, E. and R. McCulloch (1993). Variable selection via Gibbs sampling. *Journal of the American Statistical Association 88*, 881–889.

Geweke, J. (1989). Bayesian inference in econometric models using monte carlo integration. *Econometrica 57*, 1317–1339.

Gibson, T., J. Davies, J. Crane, and A. Henry (1987, September). Knee pain in sports people–a prospective study. *British Journal of Sports Medicine 21*(3), 115–7.

Givens, G. and A. Raftery (1996). Local adaptive importance sampling for multivariate densities with strong nonlinear relationships. *Journal of the American*

Statistical Association 91, 132–141.

Glymour, C., D. Madigan, D. Pregibon, and P. Smyth (1997). Statistical themes and lessons for data mining. *Data Mining and Knowledge Discovery 1*(1), 11–28.

Good, I. J. (1965). *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*. MIT Press.

Gordon, L. and R. Olshen (1984). Almost surely consistent nonparametric regression from recursive partitioning schemes. *Journal of Multivariate Analysis 15*, 147–163.

Grambsch, P., E. Dickson, M. Kaplan, G. LeSage, T. Fleming, and A. Langworthy (1989). Extramural cross-validation of the Mayo primary biliary cirrhosis survival model establishes its generalizability. *Hepatology 10*, 846–850.

Green, P. (1987). Iteratively reweighted least squares for maximum likelihood estimation and some robust and resistant alternatives (with discussion). *Journal of the Royal Statistical Society (series B) 46*, 149–192.

Hastie, T. and R. Tibshirani (1990). *Generalized Additive Models*. Chapman and Hall.

Hoeting, J., D. Madigan, A. Raftery, and C. Volinsky (1999). Bayesian model averaging: A practical tutorial. *Statistical Science*.

Huber, P. (1964). Robust estimation of a location parameter. *Annals of Mathematical Statistics 35*, 73–101.

Intrator, O. and C. Kooperberg (1995). Trees and splines in survival analysis. *Statistical Methods in Medical Research 5*, 237–261.

Kalbfleisch, J. D. and R. L. Prentice (1980). *The Statistical Analysis of Failure Time Data*. Wiley.

Keogh, E. J. and M. J. Pazzani (1999). Learning augmented Bayesian classifiers: a comparison of distribution-based and classification-based approaches. In D. Heckerman and J. Whittaker (Eds.), *Artificial Intelligence and Statistics 99*, San Francisco, CA, pp. 225–230. Morgan Kaufmann Publishers.

Kong, A., J. Liu, and W. Wong (1994). Sequential imputation and Bayesian missing data problems. *Journal of the American Statistical Association 89*, 278–288.

Kononenko, I. (1990). Comparison of inductive and naïve Bayesian learning approaches to automatic knowledge acquisition. In B. Wielinga (Ed.), *Current trends in Knowledge Acquisition*, Amsterdam, The Netherlands. IOS Press.

Langley, P., W. Iba, and K.Thompson (1992). An analysis of Bayesian classifiers. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, San Jose, CA, USA, pp. 223–228. AAAI Press.

Le Cam, L. and G. Yang (1990). *Asymptotics in Statistics: Some Basic Concepts.* New York: Springer-Verlag.

Lepage, G. P. (1978). A new algorithm for adaptive multidimensional integration. *Journal of Computational Physics 27*, 192–203.

Lepage, G. P. (1980). VEGAS: An adaptive multidimensional integration program. Technical Report CLNS-80/447, Cornell University.

Littlestone, N. and M. Warmuth (1994). The weighted majority algorithm. *Information and Computation 108*(2), 212–261.

Loader, C. (1997, April). LOCFIT: An introduction. *Statistical Computing and Graphics Newsletter*. Available at http://cm.bell-labs.com/cm/ms-/departments/sia/project.

Madigan, D., K. Mosurski, and R. Almond (1996). Explanation in belief networks. *Journal of Computational and Graphical Statistics 6*, 160–181.

Madigan, D., A. E. Raftery, C. Volinsky, and J. Hoeting (1996). Bayesian model averaging. In *AAAI Workshop on Integrating Multiple Learned Models*, pp. 77–83. AAAI Press.

Markus, B., E. Dickson, P. Grambsch, T. Fleming, V. Mazzaferro, G. Klintmalm, R. Wiesner, D. V. Thiel, and T. Starzl (1989). Efficiency of liver transplantation in patients with primary biliary cirrhosis. *New England Journal of Medicine 320*(26), 1709–1713.

Mason, L., J. Baxter, P. Bartlett, and M. Frean (1999, May). Boosting algorithms as gradient descent in function space. Technical report, Research School of Information Sciences and Engineering, Australian National University.

McCullagh, P. and J. Nelder (1989). *Generalized Linear Models* (second ed.). Chapman and Hall.

Merz, C. and P. Murphy (1998). UCI repository of machine learning databases. http://www.ics.uci.edu/ mlearn/MLRepository.html. Irvine, CA: University of California, Department of Information and Computer Science.

Nelder, J. and R. Wedderburn (1972). Generalised linear models. *Journal of the Royal Statistical Society (Series A) 135*, 370–384.

Noble, J. and K. Erat (1980, February). In defence of the meniscus. a prospective study of 200 meniscectomy patients. *Journal of Bone and Joint Surgery – British Volume 62-B*(1), 7–11.

Oh, M. and J. Berger (1992). Adaptive importance sampling in Monte Carlo integration. *Journal of Statistical Computation and Simulation 41*, 143–168.

O'Kane, J., G. Ridgeway, and D. Madigan (1998). Statistical analysis of clinical variables to predict the outcome of surgical intervention in patients with knee complaints. In submission.

Penrose, K., A. Nelson, and A. Fisher (1985). Generalized body composition prediction equation for men using simple measurement techniques. *Medicine and Science in Sports and Exercise 7*(2), 189.

Raftery, A., D. Madigan, and C. Volinsky (1996). Accounting for model uncertainty in survival analysis improves predictive performance. In J. Bernardo, J. Berger, A. Dawid, and A. Smith (Eds.), *Bayesian Statistics*, Volume 5, pp. 323–349. Oxford University Press.

Ridgeway, G., D. Madigan, and T. Richardson (1999). Boosting methodology for regression problems. In D. Heckerman and J. Whittaker (Eds.), *Proceedings of Artificial Intelligence and Statistics '99*, pp. 152–161.

Ridgeway, G., D. Madigan, T. Richardson, and J. O'Kane (1998). Interpretable boosted naïve Bayes classification. In R. Agrawal, P. Stolorz, and G. Piatetsky-Shapiro (Eds.), *Proceedings, Fourth International Conference on Knowledge Discovery and Data Mining*, pp. 101–104.

Ripley, B. and R. Ripley (1999). Neural networks as statistical methods in survival analysis. In R. Dybowski and V. Gant (Eds.), *Artificial Neural Networks: Processors for Medicine*, Volume 44. Landes Biosciences Publishers.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning 5*(2), 197–227.

Schapire, R. E. (1999a). A brief introduction to boosting. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*. AAAI Press.

Schapire, R. E. (1999b). Theoretical views of boosting. In *Computational Learning Theory: Fourth European Conference, EuroCOLT'99*, pp. 1–10.

Schapire, R. E., Y. Freund, P. Bartlett, and W. S. Lee (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statis-*

tics 26(5), 1651–1686.

Schapire, R. E. and Y. Singer (1998). Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory.*

Segal, M. (1988). Regression trees for censored data. *Biometrics 44*, 35–47.

Spiegelhalter, D. (1986). Probabilistic prediction in patient management and clinical trials. *Statistics in Medicine 5*, 421–433.

Spiegelhalter, D. and R. Cowell (1992). Learning in probabilistic expert systems. In J. Bernardo, J. Berger, A. Dawid, and A. Smith (Eds.), *Bayesian Statistics*, Volume 4, pp. 447–466. Clarendon Press, Oxford.

Spiegelhalter, D. and R. Knill-Jones (1984). Statistical and knowledge-based approaches to clinical decision-support systems, with an application in gastroenterology (with discussion). *Journal of the Royal Statistical Society (Series A) 147*, 35–77.

Spiegelhalter, D., A. Thomas, N. Best, and W. Gilks (1999, June). Bugs: Bayesian inference using Gibbs sampling, version 0.60. Technical report, MRC Biostatistics Unit, Cambridge.

Swartout, W. (1983). XPLAIN: A system for creating and explaining expert consulting programs. *Artificial Intelligence 21*, 285–325.

Tibshirani, R. and K. Knight (1995, November). Model search and inference by bootstrap "bumping". Technical report, Stanford University. To appear in the Journal of Computational and Graphical Statistics.

West, M. (1992). Modelling with mixtures. In J. Bernardo, J. Berger, A. Dawid, and A. Smith (Eds.), *Bayesian Statistics*, Volume 4. Clarendon Press, Oxford.

West, M. (1993). Approximating posterior distributions by mixtures. *Journal of the Royal Statistical Society, Series B 55*, 409–422.

Whitehead, J. (1980). Fitting Cox's regression model to survival data using GLIM. *Applied Statistics 29*(3), 268–275.

Wolpert, D. (1992). Stacked generalization. *Neural Networks 5*, 241–259.

Yates, J. (1982). External correspondence: decomposition of the mean probability score. *Organisational Behaviour and Human Performance 30*, 132–156.

Zheng, Z. and G. Webb (1998). Lazy Bayesian rules. Technical Report C98/17, School of Computing and Mathematics, Deakin University, Geelong, Victoria, Australia.

Appendix A

VARIANCE OF THE IMPORTANCE SAMPLING WEIGHTS

Lemma A.1 (multivariate square completion)

$$
\begin{aligned}
& c_1(\underline{x}-\underline{a}_1)^t \underline{A}\,(\underline{x}-\underline{a}_1) + c_2(\underline{x}-\underline{a}_2)^t \underline{A}\,(\underline{x}-\underline{a}_2) = \\
&\quad \left(\underline{x}-\frac{c_1\underline{a}_1+c_2\underline{a}_2}{c_1+c_2}\right)^t (c_1+c_2)\,\underline{A}\left(\underline{x}-\frac{c_1\underline{a}_1+c_2\underline{a}_2}{c_1+c_2}\right) + \\
&\quad c_1\underline{a}_1{}^t\underline{A}\,\underline{a}_1 + c_2\underline{a}_2{}^t\underline{A}\,\underline{a}_2 - \frac{1}{c_1+c_2}(c_1\underline{a}_1+c_2\underline{a}_2)^t\underline{A}\,(c_1\underline{a}_1+c_2\underline{a}_2)
\end{aligned}
$$

Proof of Lemma A.1:

$$
\begin{aligned}
& c_1(\underline{x}-\underline{a}_1)^t \underline{A}\,(\underline{x}-\underline{a}_1) + c_2(\underline{x}-\underline{a}_2)^t \underline{A}\,(\underline{x}-\underline{a}_2) \\
= \;& c_1\underline{x}^t\underline{A}\,\underline{x} - c_1\underline{x}^t\underline{A}\,\underline{a}_1 - c_1\underline{a}_1^t\underline{A}\,\underline{x} + c_1\underline{a}_1^t\underline{A}\,\underline{a}_1 + \\
& c_2\underline{x}^t\underline{A}\,\underline{x} - c_2\underline{x}^t\underline{A}\,\underline{a}_2 - c_2\underline{a}_2^t\underline{A}\,\underline{x} + c_1\underline{a}_2^t\underline{A}\,\underline{a}_2 \\
= \;& (c_1+c_2)\,\underline{x}^t\underline{A}\,\underline{x} - \underline{x}^t\underline{A}\,(c_1\underline{a}_1+c_2\underline{a}_2) - \\
& (c_1\underline{a}_1+c_2\underline{a}_2)^t\underline{A}\,\underline{x} + c_1\underline{a}_1^t\underline{A}\,\underline{a}_1 + c_2\underline{a}_2^t\underline{A}\,\underline{a}_2 \\
= \;& (c_1+c_2)\left(\underline{x}^t\underline{A}\,\underline{x} - \underline{x}^t\underline{A}\frac{c_1\underline{a}_1+c_2\underline{a}_2}{c_1+c_2} - \left(\frac{c_1\underline{a}_1+c_2\underline{a}_2}{c_1+c_2}\right)^t\underline{A}\,\underline{x}\right) + \\
& c_1\underline{a}_1^t\underline{A}\,\underline{a}_1 + c_2\underline{a}_2^t\underline{A}\,\underline{a}_2 \\
= \;& \left(\underline{x}-\frac{c_1\underline{a}_1+c_2\underline{a}_2}{c_1+c_2}\right)^t (c_1+c_2)\,\underline{A}\left(\underline{x}-\frac{c_1\underline{a}_1+c_2\underline{a}_2}{c_1+c_2}\right) + \\
& c_1\underline{a}_1{}^t\underline{A}\,\underline{a}_1 + c_2\underline{a}_2{}^t\underline{A}\,\underline{a}_2 - \frac{1}{c_1+c_2}(c_1\underline{a}_1+c_2\underline{a}_2)^t\underline{A}\,(c_1\underline{a}_1+c_2\underline{a}_2)
\end{aligned}
$$

Lemma A.1 generalizes in the obvious way when adding more than two quadratic forms.

□

Lemma A.2 *If, for $i = 1, \ldots, n$, $y_i|\mu, \Sigma \sim N_d(\mu, \Sigma)$ with known Σ and $\mu \sim N_d(\mu_0, \Lambda_0)$ then*

$$\mu|y, \Sigma \sim N_d(\mu_n, \Lambda_n)$$

where

$$\begin{aligned} \mu_n &= \left(\Lambda_0^{-1} + n\Sigma^{-1}\right)^{-1}(\Lambda_0^{-1}\mu_0 + n\Sigma^{-1}\overline{y}) \\ \Lambda_n^{-1} &= \Lambda_0^{-1} + n\Sigma^{-1} \end{aligned}$$

Proof: Discussion of Lemma A.2 is in Gelman, Carlin, Stern, and Rubin (1995).

□

For the main result, we will assume a non-informative prior for μ. That is, we will assume that Λ_0 takes on some value λ for the diagonal elements and 0 for the off-diagonal elements. Λ_0^{-1} then is also a diagonal matrix with the value $\frac{1}{\lambda}$ on the diagonal. A non-informative prior results when λ becomes arbitrarily large and so Λ_0^{-1} becomes the 0 matrix. The posterior simplifies under this prior to

$$\mu|y, \Sigma \sim N_d\left(\overline{y}, \frac{1}{n}\Sigma\right). \tag{A.1}$$

Theorem A.1 (Variance of the importance sampling weights) *If, for $i = 1, \cdots, n$,*

1. *$y_i|\mu, \Sigma \sim N_d(\mu, \Sigma)$ with known Σ,*

2. *$g_1 = \{y_1, \cdots, y_{n_1}\}$ and $g_2 = \{y_{n_1+1}, \cdots, y_{n_1+n_2}\}$, and*

3. *$\mu \sim N_d(\mu_0, \Lambda_0)$*

then

$$\lim_{\Lambda_0^{-1} \to 0} E_{g_2} E_{g_1} Var_{\mu|g_1,\Sigma}\left(\frac{\phi(\mu|g_1, g_2, \Sigma)}{\phi(\mu|g_1, \Sigma)}\right) = \left(\frac{n_1 + n_2}{n_1}\right)^d - 1 \tag{A.2}$$

Proof of Theorem A.1:

This is a multivariate result. In the statement of the theorem and in the proof y_i, μ, S_1, and S_2 are d-dimensional vectors and Σ is $d \times d$ matrix. The limit implies that the prior for μ is non-informative so we can assume the posterior as specified in A.1.

The expectation of the weights are trivially 1.

$$\begin{aligned} E_{\mu|g_1,\Sigma}\frac{\phi(\mu|g_1,g_2,\Sigma)}{\phi(\mu|g_1,\Sigma)} &= \int_{-\infty}^{\infty}\frac{\phi(\mu|g_1,g_2,\Sigma)}{\phi(\mu|g_1,\Sigma)}\phi(\mu|g_1,\Sigma)d\mu \\ &= \int_{-\infty}^{\infty}\phi(\mu|g_1,g_2,\Sigma)d\mu \\ &= 1 \end{aligned}$$

The expectation of the square of the weights follow. Let $S_1 = \sum\limits_{y_i \in g_1} y_i$ and $S_2 = \sum\limits_{y_i \in g_2} y_i$.

$$E_{g_2}E_{g_1}E_{\mu|g_1,\Sigma}\left(\frac{\phi(\mu|g_1,g_2,\Sigma)}{\phi(\mu|g_1,\Sigma)}\right)^2$$

$$= E_{S_1,S_2}E_{\mu|g_1,\Sigma}\left(\frac{\left|\frac{1}{n_1+n_2}\Sigma\right|^{-\frac{1}{2}}(2\pi)^{-\frac{d}{2}}\exp\left(-\frac{1}{2}\left(\mu-\frac{S_1+S_2}{n_1+n_2}\right)^t\left(\frac{1}{n_1+n_2}\Sigma\right)^{-1}\left(\mu-\frac{S_1+S_2}{n_1+n_2}\right)\right)}{\left|\frac{1}{n_1}\Sigma\right|^{-\frac{1}{2}}(2\pi)^{-\frac{d}{2}}\exp\left(-\frac{1}{2}\left(\mu-\frac{S_1}{n_1}\right)^t\left(\frac{1}{n_1}\Sigma\right)^{-1}\left(\mu-\frac{S_1}{n_1}\right)\right)}\right)^2$$

$$\begin{aligned} = \left(\frac{n_1+n_2}{n_1}\right)^d E_{S_1,S_2}\int_{-\infty}^{\infty}\exp\Bigg[-\frac{1}{2}\Bigg(&2(n_1+n_2)\left(\mu-\frac{S_1+S_2}{n_1+n_2}\right)^t\Sigma^{-1}\left(\mu-\frac{S_1+S_2}{n_1+n_2}\right) \\ &-2n_1\left(\mu-\frac{S_1}{n_1}\right)^t\Sigma^{-1}\left(\mu-\frac{S_1}{n_1}\right)\Bigg)\Bigg] \\ &\left|\frac{1}{n_1}\Sigma\right|^{-\frac{1}{2}}(2\pi)^{-\frac{d}{2}}\exp\left[-\frac{1}{2}\left(\mu-\frac{S_1}{n_1}\right)^t\left(\frac{1}{n_1}\Sigma\right)^{-1}\left(\mu-\frac{S_1}{n_1}\right)\right]d\mu \end{aligned}$$

$$\begin{aligned} = &\left(\frac{n_1+n_2}{n_1}\right)^d\left(\frac{n_1}{2\pi}\right)^{\frac{d}{2}}|\Sigma|^{-\frac{1}{2}} \\ &E_{S_1,S_2}\int_{-\infty}^{\infty}\exp\Bigg[-\frac{1}{2}\Bigg(2(n_1+n_2)\left(\mu-\frac{S_1+S_2}{n_1+n_2}\right)^t\Sigma^{-1}\left(\mu-\frac{S_1+S_2}{n_1+n_2}\right) \\ &\qquad -n_1\left(\mu-\frac{S_1}{n_1}\right)^t\left(\frac{1}{n_1}\Sigma\right)^{-1}\left(\mu-\frac{S_1}{n_1}\right)\Bigg)\Bigg]d\mu \end{aligned}$$

Applying Lemma A.1 for the first time we can collect the terms involving μ to resemble a multivariate normal distribution with covariance $\frac{1}{n_1+2n_2}\Sigma$, which is simple to integrate.

$$= \left(\frac{n_1+n_2}{n_1}\right)^d \left(\frac{n_1}{2\pi}\right)^{\frac{d}{2}} |\Sigma|^{-\frac{1}{2}}$$
$$E_{S_1,S_2} \int_{-\infty}^{\infty} \exp\left[-\frac{1}{2}\left(\left(\mu - \frac{S_1+2S_2}{n_1+2n_2}\right)^t \left(\frac{1}{n_1+2n_2}\Sigma\right)^{-1} \left(\mu - \frac{S_1+2S_2}{n_1+2n_2}\right)\right)\right]$$
$$\exp\left[-\frac{1}{2}\left(2(n_1+n_2)\left(\frac{S_1+S_2}{n_1+n_2}\right)^t \Sigma^{-1} \left(\frac{S_1+S_2}{n_1+n_2}\right) - n_1\left(\frac{S_1}{n_1}\right)^t \Sigma^{-1} \left(\frac{S_1}{n_1}\right)\right.\right.$$
$$\left.\left. -(n_1+2n_2)\left(\frac{S_1+2S_2}{n_1+2n_2}\right)^t \Sigma^{-1} \left(\frac{S_1+2S_2}{n_1+2n_2}\right)\right)\right] d\mu$$

$$= \left(\frac{n_1+n_2}{n_1}\right)^d \left(\frac{n_1}{2\pi}\right)^{\frac{d}{2}} |\Sigma|^{-\frac{1}{2}} (2\pi)^{\frac{d}{2}} (n_1+2n_2)^{-\frac{d}{2}} |\Sigma|^{\frac{1}{2}}$$
$$E_{S_1,S_2} \exp\left[-\frac{1}{2}\left(\frac{2}{n_1+n_2}(S_1-(-S_2))^t \Sigma^{-1} (S_1-(-S_2)) - \frac{1}{n_1} {S_1}^t \Sigma^{-1} S_1\right.\right.$$
$$\left.\left. -\frac{1}{n_1+2n_2}(S_1-(-2S_2))^t \Sigma^{-1} (S_1-(-2S_2))\right)\right]$$

Since S_1 is the sum of n_1 *iid* multivariate normal random variables $S_1 \sim N_d(n_1\mu, n_1\Sigma)$.

$$= \left(\frac{n_1+n_2}{n_1}\right)^d (n_1)^{\frac{d}{2}} (n_1+2n_2)^{-\frac{d}{2}}$$
$$E_{S_2} \int_{-\infty}^{\infty} \exp\left[-\frac{1}{2}\left(\frac{2}{n_1+n_2}(S_1-(-S_2))^t \Sigma^{-1} (S_1-(-S_2)) - \frac{1}{n_1} {S_1}^t \Sigma^{-1} S_1\right.\right.$$
$$-\frac{1}{n_1+2n_2}(S_1-(-2S_2))^t \Sigma^{-1} (S_1-(-2S_2))$$
$$\left.\left. +(S_1-n_1\mu)^t (n_1\Sigma)^{-1} (S_1-n_1\mu)\right)\right] |n_1\Sigma|^{-\frac{1}{2}} (2\pi)^{-\frac{d}{2}} dS_1$$

Completing the square for the second time to collect the S_1 terms we obtain an integral proportional to a multivariate normal density with covariance $\frac{(n_1+n_2)(n_1+2n_2)}{n_1+3n_2}\Sigma$.

$$= \left(\frac{n_1+n_2}{n_1}\right)^d (n_1)^{\frac{d}{2}}(n_1+2n_2)^{-\frac{d}{2}}(n_1)^{-\frac{d}{2}}|\Sigma|^{-\frac{1}{2}}|2\pi|^{-\frac{d}{2}}$$
$$E_{S_2}\int_{-\infty}^{\infty}\exp\left[-\frac{1}{2}\left(\frac{n_1+3n_2}{(n_1+n_2)(n_1+2n_2)}(S_1-\mu^*)^t\Sigma^{-1}(S_1-\mu^*)\right)\right]$$
$$\exp\left[-\frac{1}{2}\left(\frac{2}{n_1+n_2}(-S_2)^t\Sigma^{-1}(-S_2)-\frac{1}{n_1}0^t\Sigma^{-1}0\right.\right.$$
$$-\frac{1}{n_1+2n_2}(-2S_2)^t\Sigma^{-1}(-2S_2)+\frac{1}{n_1}(n_1\mu)^t\Sigma^{-1}(n_1\mu)$$
$$-\frac{(n_1+n_2)(n_1+2n_2)}{n_1+3n_2}$$
$$\left.\left.\left(\mu-\frac{2n_2S_2}{(n_1+n_2)(n_1+2n_2)}\right)^t\Sigma^{-1}\left(\mu-\frac{2n_2S_2}{(n_1+n_2)(n_1+2n_2)}\right)\right)\right]dS_1$$

where $\mu^* = \frac{\mu(n_1^2+3n_1n_2+2n_2^2)-2n_2S_2}{n_1+3n_2}$

$$= \left(\frac{n_1+n_2}{n_1}\right)^d (n_1+2n_2)^{-\frac{d}{2}}|\Sigma|^{-\frac{1}{2}}|2\pi|^{-\frac{d}{2}}|2\pi|^{\frac{d}{2}}\left(\frac{(n_1+n_2)(n_1+2n_2)}{n_1+3n_2}\right)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}$$
$$E_{S_2}\exp\left[-\frac{1}{2}\left(-\frac{2n_1}{(n_1+n_2)(n_1+2n_2)}S_2{}^t\Sigma^{-1}S_2\right.\right.$$
$$-\frac{4n_2^2}{(n_1+n_2)(n_1+2n_2)(n_1+3n_2)}$$
$$\left(S_2-\mu\frac{(n_1+n_2)(n_1+2n_2)}{2n_2}\right)^t\Sigma^{-1}\left(S_2-\mu\frac{(n_1+n_2)(n_1+2n_2)}{2n_2}\right)$$
$$\left.\left.+n_1\mu^t\Sigma^{-1}\mu\right)\right]$$

Since S_2 is the sum of n_2 *iid* multivariate normal random variables $S_2 \sim N_d(n_2\mu, n_2\Sigma)$.

$$= \left(\frac{n_1+n_2}{n_1}\right)^d\left(\frac{n_1+n_2}{n_1+3n_2}\right)^{\frac{d}{2}}$$
$$\int_{-\infty}^{\infty}\exp\left[-\frac{1}{2}\left(-\frac{2n_1}{(n_1+n_2)(n_1+2n_2)}S_2{}^t\Sigma^{-1}S_2\right.\right.$$

$$-\frac{4n_2^2}{(n_1+n_2)(n_1+2n_2)(n_1+3n_2)}$$
$$\left(S_2-\mu\frac{(n_1+n_2)(n_1+2n_2)}{2n_2}\right)^t\Sigma^{-1}\left(S_2-\mu\frac{(n_1+n_2)(n_1+2n_2)}{2n_2}\right)$$
$$+(S_2-n_2\mu)^t(n_2\Sigma)^{-1}(S_2-n_2\mu)+n_1\mu^t\Sigma^{-1}\mu\Bigg)\Bigg]|n_2\Sigma|^{-\frac{1}{2}}(2\pi)^{-\frac{d}{2}}dS_2$$

Completing the square for the third and last time to collect the S_2 terms...

$$=\left(\frac{n_1+n_2}{n_1}\right)^d\left(\frac{n_1+n_2}{n_1+3n_2}\right)^{\frac{d}{2}}n_2{}^{-\frac{d}{2}}|\Sigma|^{-\frac{1}{2}}(2\pi)^{-\frac{d}{2}}$$
$$\int_{-\infty}^{\infty}\exp\left[-\frac{1}{2}\left(\frac{n_1+n_2}{n_2(n_1+3n_2)}(S_2-n_2\mu)^t\Sigma^{-1}(S_2-n_2\mu)\right)\right]$$
$$\exp\Bigg[-\frac{1}{2}\Bigg(-\frac{4n_2^2}{(n_1+n_2)(n_1+2n_2)(n_1+3n_2)}$$
$$\left(\mu\frac{(n_1+n_2)(n_1+2n_2)}{2n_2}\right)^t\Sigma^{-1}\left(\mu\frac{(n_1+n_2)(n_1+2n_2)}{2n_2}\right)$$
$$+\frac{1}{n_2}(n_2\mu)^t\Sigma^{-1}(n_2\mu)+n_1\mu^t\Sigma^{-1}\mu$$
$$-\frac{n_2(n_1+3n_2)}{n_1+n_2}\left(\mu\frac{n_1+n_2}{n_1+3n_2}\right)^t\Sigma^{-1}\left(\mu\frac{n_1+n_2}{n_1+3n_2}\right)\Bigg)\Bigg]dS_2$$

The first exponential in the integrand is proportional to a multivariate normal density with covariance $\frac{n_2(n_1+3n_2)}{n_1+n_2}\Sigma$. The second exponential term equals 1.

$$=\left(\frac{n_1+n_2}{n_1}\right)^d\left(\frac{n_1+n_2}{n_1+3n_2}\right)^{\frac{d}{2}}n_2{}^{-\frac{d}{2}}|\Sigma|^{-\frac{1}{2}}(2\pi)^{-\frac{d}{2}}\left(\frac{n_2(n_1+3n_2)}{n_1+n_2}\right)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}(2\pi)^{\frac{d}{2}}$$
$$=\left(\frac{n_1+n_2}{n_1}\right)^d$$

And so, as we wished to prove

$$\lim_{\Lambda_0^{-1}\to 0}E_{g_2}E_{g_1}Var_{\mu|g_1,\Sigma}\left(\frac{\phi(\mu|g_1,g_2,\Sigma)}{\phi(\mu|g_1,\Sigma)}\right)=\left(\frac{n_1+n_2}{n_1}\right)^d-1 \qquad \text{(A.3)}$$

□

As a result the effective sample size is

$$ESS=\frac{m}{1+Var(w)}\approx m\left(\frac{n_1}{\mathrm{n}_1+\mathrm{n}_2}\right)^d \qquad \text{(A.4)}$$

VITA

Greg Ridgeway was born July 9, 1973 in Annapolis, Maryland. He received his Bachelor of Science in Statistics with a concentration in Mathematics in March 1995 from the California Polytechnic State University in San Luis Obispo, California. Concurrent with his undergraduate education he worked as an data analyst from 1993 to 1995 in psychometric research at the Atascadero State Hospital, California Department of Mental Health and from 1994 to 1995 for UC Berkeley's Integrated Hardwood Range Management Program. He worked as a statistician for Microsoft Research in Redmond, Washington from 1996 to 1997 while attending graduate school. In December of 1997 he received his Master of Science from the Department of Statistics at the University of Washington. In August of 1999 he received his Doctor of Philosophy from the Department of Statistics at the University of Washington.