CLASSIFIERS OF MASSIVE AND STRUCTURED DATA PROBLEMS: ALGORITHMS AND APPLICATIONS

BY SUHRID BALAKRISHNAN

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

David Madigan

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

October, 2007

ABSTRACT OF THE DISSERTATION

Classifiers of Massive and Structured Data Problems: Algorithms and Applications

by Suhrid Balakrishnan

Dissertation Director: David Madigan

The last two decades have seen the emergence of vast and unprecedented data repositories. Extraordinary opportunities now present themselves for new data analysis methods that can harness these repositories. As larger and larger amounts of widely varying types of data are constantly being collected and assimilated, the task of making use of such data opens up interesting and challenging avenues of research.

This thesis deals with specific problems in data mining and machine learning in this setting. In particular we describe algorithms and applications for classification problems where computational restrictions become limiting (resource bounded algorithms and online/streaming algorithms) as well as models and algorithms for certain problems where the structure of the input is leveraged to provide not only accurate, but also interpretable classifiers.

Acknowledgements

This thesis would not have been written without the assistance of my teachers, colleagues friends and family.

I'd like to begin by thanking David Madigan for being a perfect research advisor—I couldn't have asked for better guidance or support. I'd also like to thank my other mentors and teachers, Diane Lambert, Tapas Kanungo, Panos Georgopoulos, Marianthi Ierapetritou, Muthu Muthukrishnan, Casimir Kulikowski, Vladimir Pavolvic, Ahmed Elgammal, for all their help through the years.

I'd also like to thank my colleagues and peers from machine learning reading groups for stimulating discussions: Ofer Melnik, Alex Strehl, Dmitriy Fradkin and Chris Mesterharm.

Finally, I'd like to thank my parents and friends for their love and support through my doctoral programme. Without the company of Vasisht and Akshay or without Evi's love and support, this thesis would not have been written.

Dedication

To my family and friends

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The last two decades have seen the emergence of vast and unprecedented data repositories. Extraordinary opportunities now present themselves for new data analysis methods that can harness these repositories. The opportunities cut across a range of human endeavors such as biology, finance, retailing, and drug discovery to name a few. As larger and larger amounts of widely varying types of data are constantly being collected and assimilated, the task of making use of such data opens up interesting and challenging avenues of research.

This thesis deals with specific problems in data mining and machine learning, broad sub-fields of both artificial intelligence and statistics. In particular, the focus will be on modelling and algorithmic problems in the supervised learning setting, where the goal is to approximate/learn the behavior of a system that you only get to see examples of (regression/classification, or more abstractly, the problem of induction). The supervised setting is amenable to theoretical analysis (which informs the modelling of the complex interactions in the domain) and the range of practical applications that it encompasses ensures no lack of challenging and interesting problems.

It is known that the problem of induction is impossible in complete generality. This can be formalized in "No free lunch theorems" (Devroye et al., 1996; Wolpert, 2001) which essentially show that in order to be able to prove predictions made will be of high quality, assumptions on how the given examples (or training data) are linked to the data we'd like to predict (also called test data) are necessary. Further, additional restrictions are also required on the complexity of the phenomena we are trying to model—without this restriction, for any two learning algorithms, there would be just

as many phenomena where one algorithm is better than the other and vice-versa[1]. Even more interestingly, these theorems also prove that no such set of assumptions are universally the best (Wolpert, 1994).

Thus, much modelling effort is spent making reasonable assumptions that allow for useful/practical predictive models. Note that the final task in this supervised learning framework is always the same: predict the outcomes for new/unseen inputs as best you can, in other words, learn how to generalize well (where how well one predicts/generalizes is usually measured quantitatively using a loss function). A number of theoretical frameworks have been proposed which make distinct assumptions and thus enable a modeler to make various types of claims.

1.1 Models for Supervised Learning

This thesis will deal (mostly) with learning problems in a probabilistic framework. Here, the basic assumption is that the data are generated from some (stationary but unknown) probabilistic model. Further, it is typical to assume that the training data are generated by independently sampling this distribution (i.i.d., or independently and identically distributed samples). Further specific assumptions are made by different theories of learning which impact the kinds of claims they then imply.

Here we present sketches of just two of the main competing theories that have wide practical application—structural risk minimization (via VC theory) and Bayesian theory[2]. We will need some notation going forward, and $\mathbf{x}$ will denote inputs (say in $\mathbb{R}^n$), y will denote the outputs (in $\{-1, 1\}$—for binary classification), and let $P(\mathbf{x}, y)$ denote the distribution from which the data set[3] has been drawn $D = \{\mathbf{x}_i, y_i\}_{i=1}^t$. The distribution $P(\mathbf{x}, y)$ is unknown. A classifier is defined by the deterministic function $f(\boldsymbol{\beta}, \mathbf{x})$ where $\boldsymbol{\beta} \in B$ parametrizes the set of functions (or hypothesis class) the classifier

[1]This would in turn imply impossibility of generalization. This holds for most reasonable measures of "better"—see Wolpert (2001).

[2]Hence we will not deal with PAC (Valiant, 1984) and Statistical Physics (Wolpert, 1994) or more recent theories like PAC-Bayes (McAllester, 1999) and the online learning framework (Littlestone, 1988; Kivinen and Warmuth, 1997).

[3]Formally, this is a training data sequence (and not set). However, the term set has become standard, and we will thus use it throughout this document.

must be chosen from. How well (or poorly) a classifier does on examples is measured by the loss function $l(\boldsymbol{\beta}, \mathbf{x}_i, y_i)$, which maps the prediction $f(\boldsymbol{\beta}, \mathbf{x})$, and actual target y, to a cost or value. Typically, the range of the loss function is thus positive.

1.1.1 Structural Risk Minimization

Uniform Convergence/VC Statistical learning theory was popularized by Vapnik (Vapnik and Chervonenkis, 1971; Vapnik, 1996). This theory makes the assumptions that the dataset seen D, is drawn from the unknown distribution P in an iid fashion. Since the quality of predictions is judged by how well one performs on test examples, this can now be quantified using the notion of the risk of a classifier f (which is parametrized by $\boldsymbol{\beta} \in B$, the hypothesis class) as: $R(\boldsymbol{\beta}) = \mathbb{E}_P[l(\boldsymbol{\beta}, \mathbf{x}_i, y_i)] = \mathbb{E}_P[1_{f(\boldsymbol{\beta}, \mathbf{x}) \neq y}]$. Note that the loss used is misclassification error which takes the value 1 if the predicted label is wrong and 0 otherwise. The risk of a classifier is unknown because the distribution P is unknown. Instead only the empirical version of the risk can be evaluated, using the empirical distribution implied by the given data: $R_t(\boldsymbol{\beta}) = \frac{1}{t} \sum_{i=1}^{t} 1_{f(\boldsymbol{\beta}, \mathbf{x}_i) \neq y_i}$. In a profound result, Vapnik proved the following confidence interval style small sample bound, for any $\delta > 0$, with probability $1 - \delta$:

$$R(\boldsymbol{\beta}) \leq R_t(\boldsymbol{\beta}) + 2\sqrt{\frac{2h(\log(2t/h) + 1) + 2\log(4/\delta)}{t}}.$$

Under the assumptions, this bound allows one to estimate to desired confidence the absolute difference between the worst case risk and that of the empirical risk (Vapnik, 1998). The bound holds with probability $1-\delta$, over repeated t-sized samples (or training datasets) from the distribution P. The h term is the VC dimension of the classifier and is a formal measure of its complexity/capacity. Interpreted, the bound also inspires principles for choosing classifiers with predictive guarantees. First, in the infinite size training dataset limit ($t \to \infty$), the empirical risk $R_t(\boldsymbol{\beta})$, is the only non-zero term and a good classifier is one that minimizes this risk. This large data limit strategy is called empirical risk minimization. With small training dataset sizes, alternatively, one can choose a parameter setting such that the bound is tightest. This leads to the idea of

structural risk minimization ((Vapnik, 1996). Margin hyperplanes (as used in support vector machines) provide a concrete function class where a search over the parameter space is directly inspired by this principle and (importantly) is also computationally feasible. These hyperplanes are usually constructed in a high-dimensional feature space (a reproducing kernel Hilbert space or RKHS)—by a process which involves representing the inputs in terms of an appropriate kernel function. Moreover, extensions of this basic SVM approach (margin-maximization) have been applied to noisy data via the soft-margin SVM (Cortes and Vapnik, 1995) and these extensions have met with great empirical success[4].

1.1.2 Bayesian Theory

In this approach, the data generating model is assumed to be probabilistic. All quantities are treated as random variables, and typically one starts out with a subjective prior belief about the hypothesis class. This prior belief places a measure on all members of the hypothesis class having generated the data. Combining this prior with the probabilistic model for the data then results in the posterior belief over the hypothesis class. This combination is done via Bayes rule, using the likelihood function, which is the probability of generating the labels given the data and model parameters. In the Bayesian paradigm, this posterior distribution can be proved to be optimal for inference (Berger, 1985; Bernardo and Smith, 1994). In other words, the posterior distribution over the parameters provides the minimal loss: $E_P[l|D]$.

Hence, as opposed to other approaches in machine learning, (optimal) prediction in the Bayesian paradigm (using the Bayes classifier, or the "fully" Bayes classifier) results by integration over the posterior distribution of the model parameters.

As a large fraction of this thesis will deal with supervised learning problems in this Bayesian setting, we explain the concepts sketched in the previous paragraphs in more detail now. Mathematically, starting with a hypothesis class B, the parameters $\boldsymbol{\beta} \in B$ specify a formal probability model of the observed data D, $p(D|\boldsymbol{\beta})$. For classification,

[4]Note that although these approaches are motivated by the SVM formulation, theoretical guarantees for these extensions are not as well studied.

typically $P(y = 1|\boldsymbol{\beta}, \mathbf{x}) = f(\boldsymbol{\beta}, \mathbf{x})$ which results in a probabilistic classifier parametrized by $\boldsymbol{\beta}$. This probability $P(y = 1|\boldsymbol{\beta}, \mathbf{x})$, can then be thresholded appropriately to get the final classification[5]. Under the assumption that the data is sampled i.i.d. from P, $p(D|\boldsymbol{\beta}) = \prod_{i=1}^{t} P(y_i|\boldsymbol{\beta}, \mathbf{x}_i)$.

Assuming a prior probability distribution over the parameter space, $p(\boldsymbol{\beta})$, all information about the parameters after seeing the data D is contained in the posterior $p(\boldsymbol{\beta}|D)$ and it can be evaluated using Bayes rule as follows:

$$p(\boldsymbol{\beta}|D) = \frac{P(\boldsymbol{\beta})P(D|\boldsymbol{\beta})}{\int_B P(\boldsymbol{\beta})P(D|\boldsymbol{\beta})}.$$

The term $P(D|\boldsymbol{\beta})$ is the likelihood function and for a fixed $\boldsymbol{\beta}$ evaluates the probability of observing the given dataset using the model[6]. The term in the denominator is a normalizing constant (does not depend on $\boldsymbol{\beta}$) called the marginal data likelihood (or evidence) and is sometimes useful in Bayesian model selection (see Chapter for and example of its use).

As the formulation makes clear, for complex models and prior beliefs, the above computations are (typically) intractable, and a host of algorithms, approximations, and computational techniques have been developed. Also, approximations are also common at the prediction stage (alternatives to using the fully Bayes classifier). For example, the MAP-Bayes classifier uses just the (single) set of parameters associated with the highest posterior mass for prediction (that is, prediction using the posterior mode). Depending on the models being used, this can result in great computational savings and this approach has enjoyed much empirical success[7].

While the Bayesian framework presented above is theoretically sound and has found

---

[5]The threshold will depend on misclassification costs, which unless otherwise stated will be assumed equal for this thesis. Thus the decision rule is simply: $y = 1$, if $P(y = 1|\boldsymbol{\beta}, \mathbf{x}) \geq 0.5$ and $y = -1$ otherwise.

[6]Note that the likelihood function is not a probability distribution as it is a function of $\boldsymbol{\beta}$.

[7]The Gibbs classifier, which is a non-deterministic approximation is another alternative that avoids integration over the posterior. Here one samples a parameter setting from the posterior distribution and uses this set of parameters to makes the prediction.

widespread practical use, challenging research problems arise when applying it to complex models or data. For example, learning even the most simple classifiers in the presence of huge amounts of data (or adapting an existing classifier to an unbounded number of incoming examples) is a very challenging task and one that is extremely important practically.

This thesis covers select algorithms and applications for classification problems where computational restrictions become limiting (resource bounded algorithms and online/streaming algorithms) as well as models and algorithms for certain problems where the structure of the input is leveraged to provide not only accurate, but also interpretable classifiers.

1.2 Outline of This Document

This thesis is arranged as follows: the next three chapters deal with Bayesian classifiers. Chapters two and three describe algorithms for classifiers where the input data is massive and Chapter four describes a structured data classification problem. Chapter five then outlines a non-Bayesian approach to the same structured data problem (a non-parametric classifier). Finally we present conclusions. Each chapter is self contained. To aid the reader, individual chapter summaries follow.

1.2.1 Chapter 2: Fully Bayesian Classifiers for Small d

In the massive data scenario, even routine tasks in Bayesian analysis seem impossible. Consider evaluating a posterior distribution for the parameters of a simple (generalized) linear regression model. If the training dataset available is enormous (a large collection of input-output pairs) even this task becomes a computational nightmare. Every likelihood calculation involves examining the entire dataset which can be prohibitive if the data doesn't fit in main memory (and thus requires disk access). This automatically rules out standard approaches like Markov Chain Monte Carlo (MCMC). This chapter deals with a one-pass (or online/streaming) algorithm for solving this problem for general models when the numbers of parameters are small—medium. Data items

are examined sequentially and a representation of the posterior distribution continually maintained. The main techniques used are sequential Monte Carlo (particle filtering) methods with a novel resampling technique based on kernel smoothing Balakrishnan and Madigan (2006a). We show asymptotic convergence for the method and present empirical results.

1.2.2 Chapter 3: Sparse linear classifiers for Large d

Predictive models for text classification (categorizing news stories as related to finance or not, for example) typically involve extremely high-dimensional feature spaces where the inputs are also very sparse. Hundreds of thousands of "bag-of-word" features are not uncommon and typically only a few words occur in each document. For classifiers to be practical in this setting, feature selection is necessary, and the combined feature selection and shrinkage method using L_1 regularization (the Lasso, Tibshirani (1996)) is an attractive option with proven empirical performance. Unfortunately, the size of the resulting optimization problem and massive/unbounded numbers of input examples necessitate alternative approaches to solve this problem. This chapter deals with algorithms for learning such sparse linear classifiers that have bounded memory costs (independent of the number of examples in the dataset), exploit any inherent sparsity of the data, and are sequential in nature. There is an online (one-pass) version of the algorithm as well. These algorithms are theoretically sound and shown to be practical, accurate and efficient Balakrishnan and Madigan (2006b). The main techniques used are a simple quadratic approximation to the likelihood of the parameters, combined with careful analysis and sequential caching of the datasets' sufficient statistics.

1.2.3 Chapter 4: Finding Predictive Runs for Structured Data Classification

This chapter describes an approach to create parametric classifiers for structured classification problems—which are classification problems where the input examples (or output labels) have certain structural dependencies. In particular, we assume that groups of the input covariates are meaningfully ordered, and it is important to try and use this

information in building the final classifier. For this task, we propose a new linear model based on recent extensions to the Lasso, namely the Group Lasso (Yuan and Lin (2006), where sets of variables are modelled as effects that can be zeroed out) and the Fused Lasso (Tibshirani et al. (2005), where serial correlation between coefficients is explicitly modelled). This new model (LAPS, the Lasso with Attribute Partition Search) blends the advantages of both of the above approaches, and explicitly formalizes the problem in terms of a search for groups of correlated attributes. We also propose an algorithm to efficiently find the model coefficients, and our results on applications show the model to be very good predictively, while also capable of providing insight into the structured domain it is applied on Balakrishnan and Madigan (2007).

1.2.4 Chapter 5: A Non-parametric Approach to Structured Data Classification

The structured classification problems mentioned above was also the motivation for the work described in this chapter. Here, as an alternative to the Bayesian approach, a new extension to decision trees were applied to the problem. Decision trees are a type of non-parametric classifier that have had good empirical success and are especially well known for their interpretability, which makes them ideal in this problem domain. This chapter describes "functional" decision trees which are decision trees augmented with a new type of variable split (functional splits), designed to partition based on the overall shape of assay curves (these are time series). We present the model, outline an algorithm to learn the trees and demonstrate results on both real and applied datasets.

References

S. Balakrishnan and D. Madigan. A one-pass sequential monte carlo method for bayesian analysis of massive datasets. *Bayesian Analysis*, 1(2):345–362, 2006a.

S. Balakrishnan and D. Madigan. Algorithms for sparse linear classifiers in the massive data setting. *submitted manuscript*, 2006b.

S. Balakrishnan and D. Madigan. Finding predictive runs with laps. *submitted manuscript*, 2007.

J. O. Berger. *Statistical Decision Theory and Bayesian Analysis.* Springer Verlag, 1985.

J. M. Bernardo and A. F. M. Smith. *Bayesian Theory.* Wiley, 1994.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20 (3):273–297, 1995.

L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition.* Springer, New York, 1996.

Jyrki Kivinen and Manfred K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Inf. Comput.*, 132(1):1–63, 1997. ISSN 0890-5401. doi: http://dx.doi.org/10.1006/inco.1996.2612.

Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2, 1988.

McAllester. Some PAC-bayesian theorems. *Machine Learning*, 37, 1999.

R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society*, 67(1):91 – 108, 2005.

R. J. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58(1):267–288, 1996.

Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984.

V. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, 1996.

V. Vapnik. *Statistical Learning Theory.* Wiley, 1998.

V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probab. and its Applications*, 16(2): 264–280, 1971.

D. Wolpert. The supervised learning no-free-lunch theorems. In *World conference on Soft Computing 2001*, 2001.

D. Wolpert. The relationship between the various supervised learning formalisms. In D. Wolpert, editor, *The Mathematics of Generalization*, Boston, MA, 1994. Addison-Wesley.

M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society*, 68(Series B):49 – 67, 2006.

Chapter 2

Fully Bayesian Classifiers for Small d[1]

2.1 Introduction

Routine Bayesian data analysis relies on Monte Carlo algorithms that perform thousands or even millions of laps through the data. This can preclude Bayesian analyses of massive datasets. The Bayesian approach does, however, lend itself naturally to sequential algorithms. The posterior after the ith observation becomes the prior for the $i+1$st observation, and so on. For analyses that adopt conjugate prior distributions, this can provide a simple, scalable approach for dealing with massive datasets. However, this conjugate setup describes only a small fraction of today's Bayesian applications.

In this chapter we propose a general algorithm that performs a rigorous Bayesian computation on a small, manageable portion of the dataset and then sequentially adapts those calculations with the remaining observations. The algorithm loads each of these remaining observations into memory only once yet maintains inferential fidelity.

There exists a small literature focussed on scaling up Bayesian methods to massive datasets. A number of authors have proposed large-scale Bayesian network learning algorithms, although most of this work is not actually Bayesian per se (see, for example, Friedman et al.) and none to our knowledge is one-pass. Posse (2001) presents an algorithm for large-scale Bayesian mixture modelling. DuMouchel (1999) presents an algorithm for learning a Gamma-Poisson empirical Bayes model from massive frequency tables.

Our work improves and extends the previously proposed scheme in Ridgeway and Madigan (2002) (which is essentially the same as that outlined for static models in

[1]The basic content in this chapter has appeared in "One Pass Bayesian Analysis of Massive Datasets", Balakrishnan S. and Madigan D., Bayesian Analysis, 1, 2006.

Chopin 2002a) and formulates a one-pass method of analysis.

2.2 Bayesian computation for massive datasets

The outputs of Bayesian data analyses often take the form of estimates of expectations. Specifically, we compute the expected value of the quantity of interest, $h(\theta)$, using

$$E(h(\theta)|x_1,\ldots,x_N) = \int h(\theta) f(\theta|x_1,\ldots,x_N) d\theta \tag{2.1}$$

where $f(\theta|\mathbf{x})$, is the posterior density of the parameters given the observed data. Analytic expressions for such integrals exist only in the simplest of cases leading to a de facto reliance on Monte Carlo methods. Monte Carlo integration methods sample from the posterior, $f(\theta|\mathbf{x})$, and then estimate $E(h(\theta)|x_1,\ldots,x_N)$ as $\frac{1}{M}\sum_{i=1}^{M} h(\theta_i)$ where $\theta_1,\ldots,\theta_M$ comprise a sample of M "particles" from $f(\theta|\mathbf{x})$. The law of large numbers ensures convergence:

$$\lim_{M\to\infty} \frac{1}{M}\sum_{i=1}^{M} h(\theta_i) = \int h(\theta) f(\theta|x_1,\ldots,x_N) d\theta. \tag{2.2}$$

Importance sampling methods sample from a different density, say $g(\theta)$, and take weighted averages:

$$\begin{aligned} \int h(\theta) f(\theta|x_1,\ldots,x_N) d\theta &= \int h(\theta) \frac{f(\theta|\mathbf{x})}{g(\theta)} g(\theta) d\theta & (2.3)\\ &= \lim_{M\to\infty} \frac{1}{M}\sum_{i=1}^{M} w_i h(\theta_i) & (2.4) \end{aligned}$$

where θ_i is now a draw from $g(\theta)$ and $w_i = f(\theta_i|\mathbf{x})/g(\theta_i)$, is a weight associated with a particular particle θ_i. Since the expected value of w_i under $g(\theta)$ is 1, we need only compute weights up to a constant of proportionality and then normalize, leading to:

$$\widehat{E}(h(\theta|x_1,\ldots,x_N)) = \frac{\sum_{i=1}^{M} w_i h(\theta_i)}{\sum_{i=1}^{M} w_i}. \tag{2.5}$$

Geweke (1989) provides conditions under which these estimates are asymptotically consistent.

The algorithm in Ridgeway and Madigan (2002), essentially consists of partitioning the data $\{x_1, \ldots, x_N\}$ into two pieces, a manageable portion $D_{1:n} = x_1, \ldots, x_n$ where $n \ll N$, and the remainder of the data, $D_{n+1:N} = x_{n+1}, \ldots, x_N$, and then applying importance sampling with $g(\theta) = f(\theta|x_1, \ldots, x_n)$. Now, if the observations are conditionally independent given the parameters θ, i.e. $f(x_1, \ldots, x_n|\theta) = \prod_{i=1}^{n} f(x_i|\theta)$, the corresponding importance sampling weights have a particularly simple form. Recall that our conditions imply $w_i = f(\theta_i|D_{1:N})/g(\theta_i)$, and substitute the expressions for f, g to get:

$$\begin{aligned} w_i &= \frac{f(\theta_i|D_{1:n}, D_{n+1:N})}{f(\theta_i|D_{1:n})} \\ &= \frac{f(D_{1:n}, D_{n+1:N}|\theta_i)f(\theta_i)}{f(D_{1:n}, D_{n+1:N})} \frac{f(D_{1:n})}{f(D_{1:n}|\theta_i)f(\theta_i)} \\ &\propto f(D_{n+1:N}|\theta_i) = \prod_{x_j \in D_{n+1:N}} f(x_j|\theta_i). \end{aligned} \tag{2.6}$$

This is just the likelihood of the observations in $D_{n+1:N}$ evaluated at each particle.

Unfortunately, the Monte Carlo variance of the resulting importance sampling estimates grows quickly. Since all of the terms are positive in:

$$Var(\theta|D_{1:n}) = E(Var(\theta|D_{1:n}, D_{n+1:N})) + Var(E(\theta|D_{1:n}, D_{n+1:N})), \tag{2.7}$$

the posterior variance with the additional observations in $D_{n+1:N}$ is, in expectation, smaller than the posterior variance conditioned only on $D_{1:n}$. Therefore, although the location of the sampling density should be close to the target density, its spread will most likely be wider than that of the target. As additional observations become available, $f(\theta|D_{1:n}, D_{n+1:N})$ becomes much narrower than $f(\theta|D_{1:n})$. The result of this narrowing is that the weights of many of the original draws from the sampling density approach zero and so we have few effective draws from the target density, a phenomenon also known as degeneracy of the sample. Figure 2.1 demonstrates the problem schematically. The wider density represents the sampling density $f(\theta|D_{1:n})$

that generates the particles. However, the target density, $f(\theta|D_{1:n}, D_{n+1:N})$, shown as a dashed curve, is shifted and narrower. About half of the draws from $f(\theta|D_{1:n})$ will have importance weight near zero.

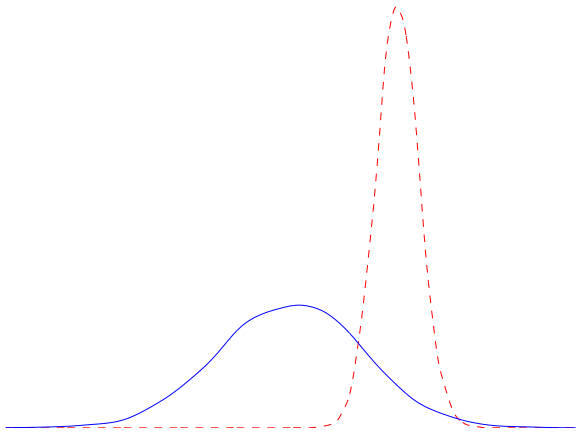


Figure 2.1: Comparison of $f(\theta|D_{1:n}, D_{n+1:N})$ (dashed) and $f(\theta|D_{1:n})$ (solid).

Ridgeway and Madigan (2002) monitor this degeneracy via the so-called effective sample size or ESS. Kong et al. (1994) provided a simple approximation for the ESS:

$$ESS = \frac{M}{1 + Var(w)} = \frac{(\sum w_i)^2}{\sum w_i^2}.$$

When the ESS drops below some pre-specified level, Ridgeway and Madigan (2002) counter the degeneracy via a resample-move or "rejuvenation" step, an idea that borrows from the particle filtering/sequential Monte Carlo literature – see, for example Doucet et al. (2001) and Gilks and Berzuini (2001). The resample-move step first resamples the particles with probabilities proportional to the weights and then applies a single Metropolis-Hastings "move" step to each particle. Each move requires a complete scan of all the data. Figure 2.2 provides an outline of the algorithm.

While the Ridgeway and Madigan (2002) scheme decreases the number of data accesses by up to 99% as compared to vanilla MCMC, it is not a one-pass algorithm since the move portion of the rejuvenation scheme, i.e. the single MCMC step, involves conditioning on all of the data processed thus far. Although Ridgeway and Madigan provide arguments that show that rejuvenations become less frequent as more data

1. Load as much data into memory as possible to form $D_{1:n}$
2. Draw M times from $f(\theta|D_{1:n})$ via Monte Carlo or Markov chain Monte Carlo
3. Iterate through the remaining observations (those that comprise $D_{n+1:N}$). For each observation, x_j, update the log-weights on all of the draws from $f(\theta|D_{1:n})$.
   Set $j = n + 1$ and $p \in (0, 1)$ to the allowable decrease in ESS
   While $j < N$
   Set $w_i = 1$, $i = 1, \ldots, M$
   While ESS $> pM$
   $j \leftarrow j + 1$
   for i in $1, \ldots, M$ do $w_i \leftarrow w_i \times f(x_j|\theta_i)$
   Resample M times with replacement from $\theta_1, \ldots, \theta_M$
   with probability proportional to w_i
   for i in $1, \ldots, M$ do one MCMC step to move θ_i
   conditioned on $n + j$ observations

Figure 2.2: The Ridgeway and Madigan (2002) algorithm for massive datasets

accumulate, this is still a significant flaw since the number of repeated data accesses per observation grows without bound.

2.3 One-pass Particle Filtering for Massive Datasets

Our proposed one-pass particle filtering algorithm 1PFS (One-pass Particle Filter with Shrinkage) differs from the Ridgeway and Madigan algorithm of Figure 2.2 only in the rejuvenation step. The new rejuvenation step uses a "shrinkage" kernel smoothing approximation to the current importance sampling distribution.

Thus 1PFS starts out with Steps 1 and 2 of the Ridgeway and Madigan (2002) algorithm (see Figure 2.2). Then, as with Ridgeway and Madigan, 1PFS iterates the outer loop of Step 3 until the ESS deteriorates below some tolerance limit (10% of M, say). Assuming that this occurs after absorbing n_1 observations, 1PFS then resamples M times with replacement the particles from the posterior conditioned on the first $n + n_1$ data points. The resample selects each particle θ_i with probability proportional to w_i.

Note that these draws still represent a sample, albeit a dependent one, from the posterior conditioned on the first $n + n_1$ data points. Several of the θ_i will appear multiple times in this new sample. For the most part this refreshed sample will be devoid of those θ_i not supported by the data.

In order to rejuvenate the sample, we propose to approximate the distribution of the resampled θ_i using kernel smoothing. Liu and West (2001) note that centering the kernels at the standard locations (i.e., at the existing particles) systematically results in a density estimate that is over-dispersed. Liu and West (2001) propose a shrinkage scheme to correct for this over-dispersion via a weighted shift of the location of the particles towards the sample mean.

More precisely, 1PFS uses kernel smoothing to approximate the importance sampling posterior density $f(\theta|D_{1:n+n_1})$ of the parameters as per:

$$\widehat{f}(\theta|D_{1:n+n_1}) \quad = \quad \sum_{i=1}^{M} K(\theta; \widetilde{\theta}_i, b^2 V) \tag{2.8}$$

where $K(\theta; s, T)$ is the value at θ of the kernel function (e.g., Gaussian) with mean s and variance matrix T. $\widetilde{\theta}_i$ and V are the shifted sample/particle values and the sample Monte Carlo variance respectively with b being the kernel bandwidth. Note that the w_i's, are all identically equal to 1 in the above formula because a resample step preceded the rejuvenation step. The shrinkage rule specifies the shifted sample locations as:

$$\widetilde{\theta}_i = a\theta_i + (1 - a)\overline{\theta} \tag{2.9}$$

where $a = \sqrt{1 - b^2}$ and $\overline{\theta}$ is the current Monte Carlo mean θ_i value. The sample drawn from the kernels placed at the shrinkage locations will not only have the correct mean (which is the original sample mean, $\overline{\theta}$ and is unchanged) but also the correct variance (the sample variance, V).

Therefore, to rejuvenate the sample, for each of the new θ_i's we simply sample from the shrinkage kernel density based approximation to the importance sample distribution

that we currently have, $\widehat{f}(\theta|D_{1:n+n_1})$. Our rejuvenated θ_i's now represent a more diverse set of parameter values with an effective sample size closer to M again. Figure 2.4 graphically walks through the resample-move process for this "smooth bootstrap" step-by-step and Figure 2.3 shows the new reweighting step to replace step 3 of the Ridgeway and Madigan (2002) algorithm.

---

3. Iterate through the remaining observations (those that comprise $D_{n+1:N}$). For each observation, x_j, update the log-weights on all of the draws from $f(\theta|D_{1:n})$. Set $j = n+1$, $p \in (0,1)$ to the allowable decrease in ESS and b, the kernel bandwidth (Note: enables computation of $a = \sqrt{1-b^2}$)
   While $j < N$
   Set $w_i = 1$, $i = 1, \ldots, M$
   While ESS $> pM$
   $j \leftarrow j+1$
   for i in $1, \ldots, M$ do $w_i \leftarrow w_i \times f(x_j|\theta_i)$
   Resample M times with replacement from $\theta_1, \ldots, \theta_M$
   with probability proportional to w_i
   Compute $\overline{\theta}$, V
   for i in $1, \ldots, M$ do
   Compute $\widetilde{\theta_i} = a\theta_i + (1-a)\overline{\theta}$
   Sample new θ_i from $K(\widetilde{\theta_i}, b^2V)$

Figure 2.3: One-pass Particle Filtering for Massive Datasets

After rejuvenating the set of θ_i, we can continue where we left off, on observation $n+n_1+1$, absorbing additional observations until either we include the entire dataset or the ESS again has dropped too low and we need to preform a new rejuvenation step.

2.3.1 Convergence of the Smooth Bootstrap; Bandwidth Selection

There exist established asymptotic (as the number of particles tends to infinity, i.e., $M \to \infty$) Central Limit Theorems for Sequential Monte Carlo methods – see Moral and Guionnet (1999), Gilks and Berzuini (2001), and Chopin (2002b). These results deal with the more general version of the sequential inference problem involving unseen state variables in addition to static model parameters. These results also apply to the

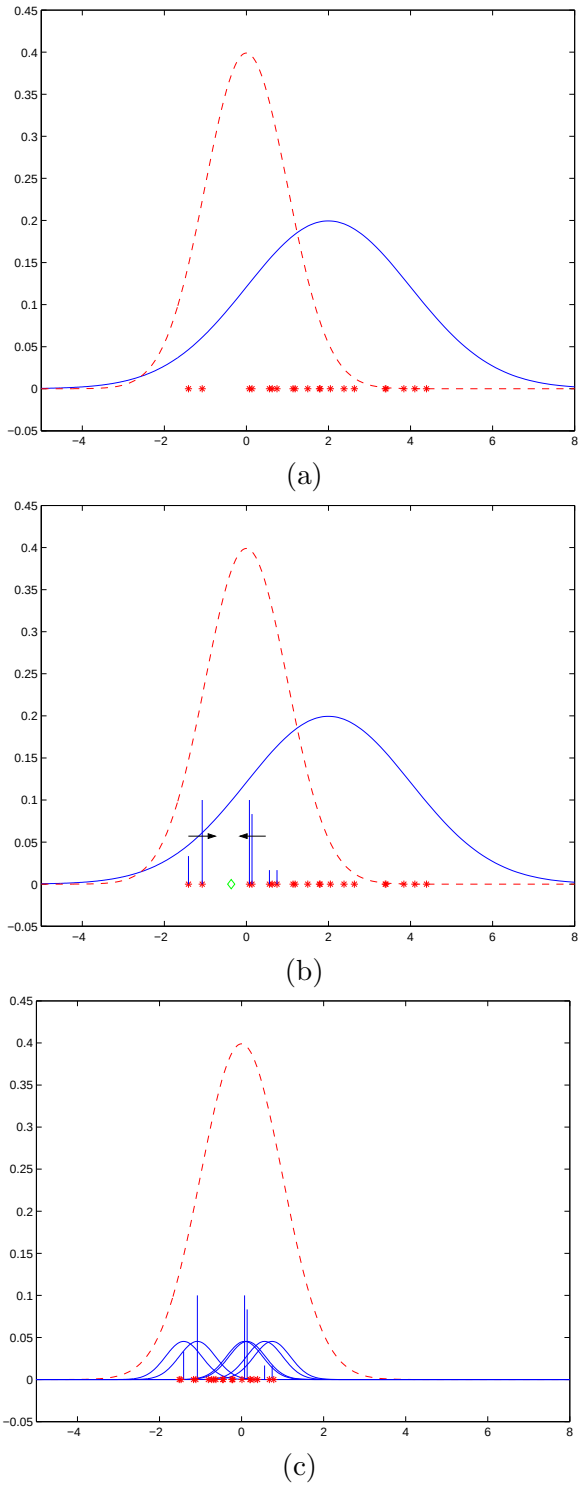


Figure 2.4: The resample-move step. (a) Generate an initial sample from $f(\theta|D_{1:n})$ (the solid curve). The stars mark the particles, the sampled θ_i. (b) Weight based on $f(\theta|D_{1:n}, D_{n+1:N})$ (the dashed density) and resample, the length of the vertical lines indicate the number of times resampled. Shrink these locations towards $\overline{\theta}$ (the open diamond) 3) For each θ_i sample from the now shifted kernel density distribution and thus diversify and obtain the new sample (the stars mark these locations).

simpler version of the problem we are concerned with, namely filtering for static model parameters. Indeed, the static-only case is better behaved and more tractable than the general problem Chopin (2002b). Further, since we don't have a general state-space model (and the sequential updating is only an artifact to reduce the number of data accesses) we are concerned solely with the convergence properties of the final posterior distribution estimate that our algorithm returns.

These Central Limit Theorems hold true for sequential Monte Carlo methods involving sampling-importance resampling and MCMC (rejuvenation) moves as per Ridgeway and Madigan. 1PFS, however, employs an "extra" approximation step involving the kernel smoothing approximation to the importance sampling posterior distribution, $\widehat{f}(\theta|x)$. Intuitively, this extra approximation will be inconsequential if, in the limit as the number of particles tend to infinity, the particles sampled from the kernel smoothed approximation to the posterior distribution will resemble random samples from the importance sampling posterior $f(\theta|x)$.

Stavropoulos and Titterington (2001) prove a restricted version of the above statement formally. Their theorem states:

Theorem 2.3.1 *Under mild conditions, for univariate θ and the Normal kernel K, the cumulative distribution function of the values generated by the kernel approximation to the posterior distribution $\widehat{f}(\theta|x)$, converges to that of the target density, $f(\theta|x)$.*

The proof has an easy multivariate generalization and can be adapted for non-Normal K as well. The assumptions under which theorem 2.3.1 holds are fairly mild, essentially the same as those required by Geweke (1989) for the importance sample estimates to converge, with the additional requirement that the kernel functions variance should shrink to zero as the number of particles tends to infinity.

For Normal kernels (where $K(s,T) = \varphi(s,T)$, the Gaussian density function), kernel density estimation literature Silverman (1986) suggests a choice of $T = V{b_M}^2$, with

$$b_M = \left(\frac{4}{(d+2)M}\right)^{\frac{1}{d+4}} \tag{2.10}$$

where d is the dimensionality of the samples, M is the number of samples and V is the sample Monte Carlo variance estimate. This choice of bandwidth is asymptotically optimal if the density being approximated is multivariate-Normal, and the samples had been obtained from this distribution Stavropoulos and Titterington (2001). In the next section we also provide empirical data supporting its use as reasonable even for small sample sizes.

2.3.2 Empirical Justification of Bandwidth Selection Rule

As stated previously, the bandwidth selection rule in Equation 2.10 is (on average) an optimal finite sample size bandwidth selection rule (minimizes the AIMSE) when the distribution being estimated is known to be multivariate Normal and we are using a Gaussian kernel. Since the addition of observations forces the posterior distribution to asymptotically approach a multivariate Normal, and as we know that for a Gaussian kernel, no other bandwidth selection rule will behave better uniformly (because then it would also be better on average) we believe the choice of bandwidth to be very reasonable (and indeed continually improving as more data is assimilated).

In order to empirically validate this claim, we performed 10 simulations of 1PFS with parameters exactly as specified previously, except for the bandwidth parameter, which we varied uniformly in the [0-1] interval in increments of 0.1. We then calculated the MSE error between the mean posterior parameter values obtained from regular MCMC and those obtained from the use of 1PFS. The following table (Table 2.1) shows the mean error (averaged over the 10 runs per bandwidth) and associated standard deviations.

The figure shows the same data on a plot. The results of our limited number of simulations point to the fact that while the bandwidth specified per Equation 10, $b_M = 0.4557$, doesn't give the lowest error (that appears to be at $b_M = 0.6$), it still clearly does provide a reasonable choice.

The following sections present two examples (both of which were previously analyzed in Ridgeway and Madigan 2002) that elucidate the application of the proposed algorithm in practice, followed by a discussion of the method and conclusions.

Bandwidth (b_M)	Mean MSE	STD MSE
0.10	0.0436	0.0169
0.20	0.0151	0.0029
0.30	0.0074	0.0012
0.40	0.0063	0.0007
0.50	0.0062	0.0008
0.60	0.0057	0.0008
0.70	0.0062	0.0007
0.80	0.0064	0.0006
0.90	0.0065	0.0008

Table 2.1: The effect of bandwidth b_M on the mean MSE error estimates (and standard deviations) obtained between regular MCMC and 1PFS for Bayesian logistic regression analysis of the outpic data.

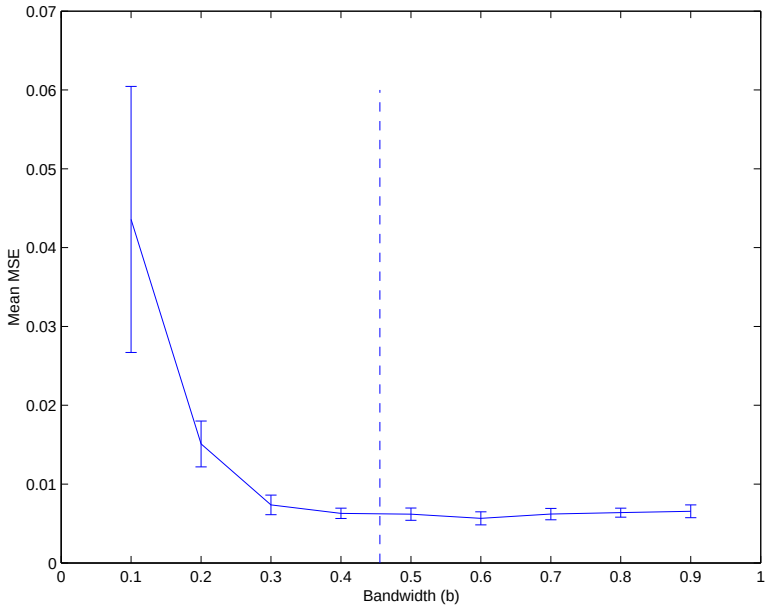


Figure 2.5: Plot of the MSE and associated standard deviation of Table 2.1. The dashed vertical line indicates the bandwidth choice recommended by Equation 10.

2.4 Example I - Fully Bayes Logistic Regression

The first example we consider concerns Bayesian logistic regression. The training data comprise vectors $\mathbf{x_i} = [x_{i_1}, \ldots, x_{i_d}]^T$ in $\mathbf{R^d}$ and $y_i \in \{0, 1\}$, $i = 1, \ldots, N$. We consider a model of the form:

$$p(y = 1|\mathbf{x}) = \psi(\boldsymbol{\beta}^T\mathbf{x}) \tag{2.11}$$

where $\boldsymbol{\beta}$ is a vector of regression coefficients and $\psi(\cdot)$ is the logistic link function. Following some recent literature on so-called "sparse" models (Tibshirani 1995; Figueiredo 2001) we use an independent Laplace prior for each component of $\boldsymbol{\beta}$:

$$\pi(\beta_i|\gamma) = \frac{\gamma}{2}e^{-\gamma|\beta_i|}, \gamma > 0, i = 1, \ldots, d.$$

This prior typically results in posterior modes of zero for many parameters and as such accomplishes simultaneous shrinkage and variable selection. Our interest here however is not in obtaining the posterior mode (see Genkin et al. 2004) but rather in fully Bayesian inference for arbitrary characteristics of the posterior distribution of $\boldsymbol{\beta}$. In the example below we set $\gamma = 5$; Genkin et al. (2004) discuss approaches for selecting γ.

Following Ridgeway and Madigan (2002) we report fully Bayesian logistic regression analysis of the "outpic data" comprising 744,963 customer records, 57 Mb when stored in double precision. Thus, in our notation, $N = 744,963$. These data originated in a major telecommunications company. The binary response variable identifies customers who have switched to a competitor. There are seven predictor variables. Five of these are continuous and two are 3-level categorical variables. Thus for logistic regression there are $d = 10$ parameters. This dataset is small enough that regular MCMC to compute $f(\boldsymbol{\beta}|D_{1:N})$, while cumbersome, is still feasible. We also used MCMC to generate the initial particles from $f(\boldsymbol{\beta}|D_{1:n})$. In both cases we used a straightforward Metropolis-within-Gibbs sampler. Ridgeway and Madigan (2002) describe this sampler in detail. The key point is that the MCMC sampler requires one complete pass through the data per iteration.

1PFS used a Gaussian kernel function. Formula 2.10 defines the kernel bandwidth. Conditioning on the first 10,000 observations (i.e., $n = 10,000$), we generated 25,000 initial particles using the MCMC algorithm, dropping the first 5,000 (i.e. $M = 20,000$). Thus we accessed each of the first 10,000 observations 25,000 times. 1PFS executed a rejuvenation step whenever the ESS dropped below 10,000 (which occurred 51 times until the whole dataset was processed) corresponding to the tolerance limit $p = 0.5$. By construction, 1PFS accessed observations 10,001-744,963 just once. Ridgeway and Madigan's algorithm with the same rejuvenation schedule also accessed the first 10,000 observations 25,000 times each, but then accessed the remaining 734,963 observations a total of 2,352,460 times or 3.2 times per observation.

In addition to 1PFS, we also fit the logistic regression model using maximum likelihood and using a full MCMC run on the entire dataset. Table 2.2 shows the resulting estimates.

	$\beta(1)$	$\beta(2)$	$\beta(3)$	$\beta(4)$	$\beta(5)$	$\beta(6)$	$\beta(7)$	$\beta(8)$	$\beta(9)$	$\beta(10)$
MLE	-0.574	0.155	0.056	0.220	-0.087	0.361	-0.358	-0.204	0.079	0.079
MCMC	-0.574	0.155	0.056	0.220	-0.087	0.360	-0.358	-0.204	0.080	0.078
1PFS	-0.574	0.156	0.056	0.221	-0.087	0.360	-0.357	-0.204	0.079	0.079

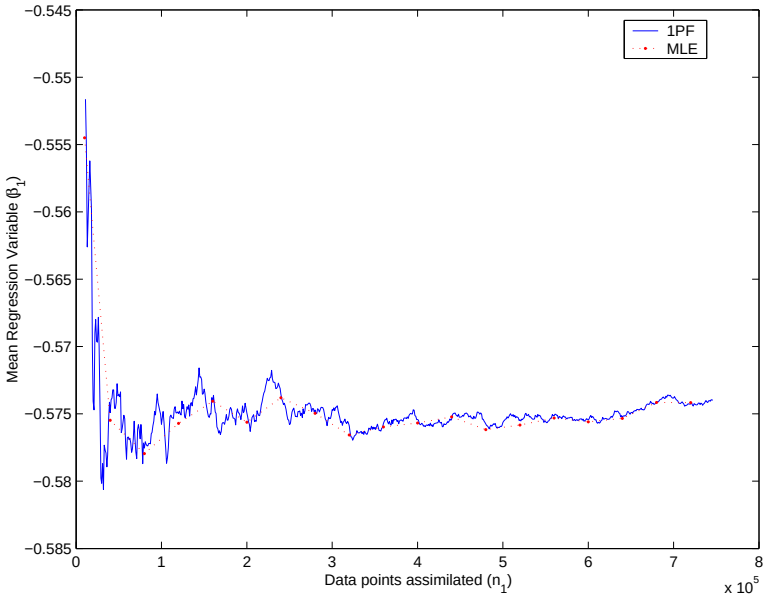
Table 2.2: Mean $\boldsymbol{\beta}$ estimates obtained from Bayesian logistic regression analysis of the outpic data.

Table 2.3 shows the total number of data accesses.

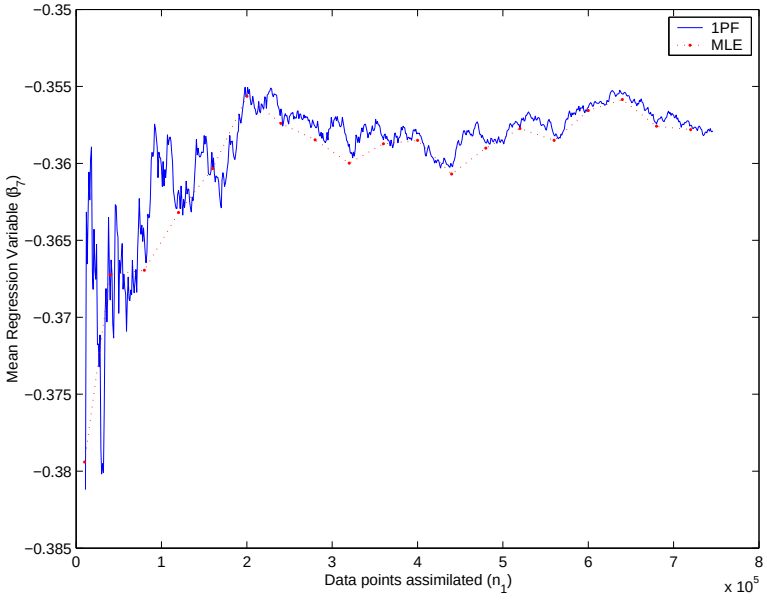
Algorithm	first 10,000	next 734,963
MCMC	2.5×10^8	1.8×10^{10}
R&M	2.5×10^8	2.4×10^6
1PFS	2.5×10^8	7.3×10^5

Table 2.3: Total number of data accesses for MCMC (Markov chain Monte Carlo on the entire dataset), R&M (Ridgeway and Madigan's Particle Filter), and 1PFS (One-pass Kernel-based Particle Filter).

1PFS not only scans (most of) the dataset only once but also produces parameter estimates that are very close to the desired values. Indeed, on smaller subsets of the data, the maximum likelihood estimates for the parameter values and those obtained via 1PFS are very similar as well (Figure 2.6).



(a)



(b)

Figure 2.6: Plot showing the representative mean posterior parameter values ($\beta(1)$ and $\beta(7)$) determined via 1PFS as a function of the amount of data processed. Also shown on the plot is the corresponding MLE for the same amount of data.

2.5 Example II - Mixtures of Transition Models

Mixtures of first-order transition models (i.e., finite state Markov chains) have attracted recent attention in a variety of applications such as web user modelling (Cadez et al. 2000; Ridgeway 1997) and unsupervised training of robots (Ramoni et al.). This mixture model assumes that the data comprise N state sequences of random length and that one of C transition matrices generated each sequence. However, neither the transition matrices nor the mixing proportions are known. Thus the unknown parameters of this model are the C transition matrices (each $S \times S$, where S is the size of the state space), $P_1, \ldots, P_C$, the mixing vector of length C, and the N cluster assignments, $z_j \in \{1, \ldots, C\}, j = 1, \ldots, N$. A Bayesian analysis estimates the posterior distribution of these unknowns given a set of observed sequences. We assume that both C and S are fixed.

Ridgeway and Madigan (2002) describe a simple-to-implement Gibbs sampler that generates draws from this posterior distribution. However, each iteration requires two scans of the entire dataset, one for the matrix update and one for the cluster assignment update. Consequently, computation time becomes prohibitive for any N much bigger than a few tens of thousands. Ridgeway and Madigan (2002) propose a particle filtering algorithm for this model and, as with the logistic regression example, the number of data accesses decreases dramatically compared to the Gibbs sampler. However each rejuvenation step requires a complete scan of the data to that point. Here we apply the 1PFS algorithm to this model in the context of a particular example.

Specifically we generated $N = 1$ million sequences of length between 5 and 20 from two 4×4 transition matrices. We used the first $n = 1000$ sequences to obtain the initial sample of $M = 1000$ particles. For this example 1PFS executes a rejuvenation step each time ESS drops below 100 (i.e. $p = 0.1$). Because both the rows of the transition matrices and the vector of mixing proportions must sum to one, the kernel function, $K(.,.)$, we chose for this example was Dirichlet. Following Aitchison and Lauder (1985), we choose the bandwidth b that maximizes the pseudo-likelihood (the average leave-one-out cross validation approximated likelihood).

The use of the Dirichlet kernel involves one extra detail. The Liu and West (2001) shrinkage rule requires a parametrization of the kernel $K(\widetilde{\theta}_i, b^2V)$ in terms of its mean $\widetilde{\theta}_i$ and variance b^2V. Unfortunately, starting from a mean and variance, a closed-form expression for the corresponding Dirichlet distribution Dirichlet(α) does not exist. Following Ronning (1989) we compute an approximation to α by matching first and second moments. Specifically, the parameter values α_{isc} for the i^{th} row of the transition matrix P_c are:

$$\alpha_{isc} = \widetilde{\theta}_{isc} \sum_s \alpha_{isc} \tag{2.12}$$

$$\log \sum_s \alpha_{isc} = \frac{1}{S-1} \sum_{s=1}^{S-1} \log \left(\frac{\widetilde{\theta}_{isc}(1-\widetilde{\theta}_{isc})}{b^2 V_{isc}} - 1 \right) \tag{2.13}$$

Here we model each row independently. A similar set of equations exists for the mixing vector's parameters (implying a total of $d = 25$ independent parameters).

Except for the first 1000 observations, which generate the initial set of particles, 1PFS accesses each of the remaining observations once. Once again, this represents a substantial computational savings as compared to the Ridgeway and Madigan (2002) scheme. A Gibbs sampler, conditioned on the entire dataset, would need to access each observation 2000 times.

While efficiency as measured by the number of data accesses is important in the analysis of massive datasets, precision of parameter estimates is also important. Figure 2.7 shows the marginal posterior densities for the 16 transition probabilities from the first mixture component's transition matrix. The smooth density plot is based on the $M = 1000$ 1PFS particles. The figure also marks the location of the parameter value that generated the data. All of these values are within the region with most of the posterior mass.

With 1,000,000 observations, a Gibbs sampler here is prohibitively expensive (scaling computational time linearly and disregarding memory constraints leads to an estimate of around 2 months CPU time). On a subset of the data comprising the first 10,000 observations, the two methods produced nearly identical posterior distributions – see

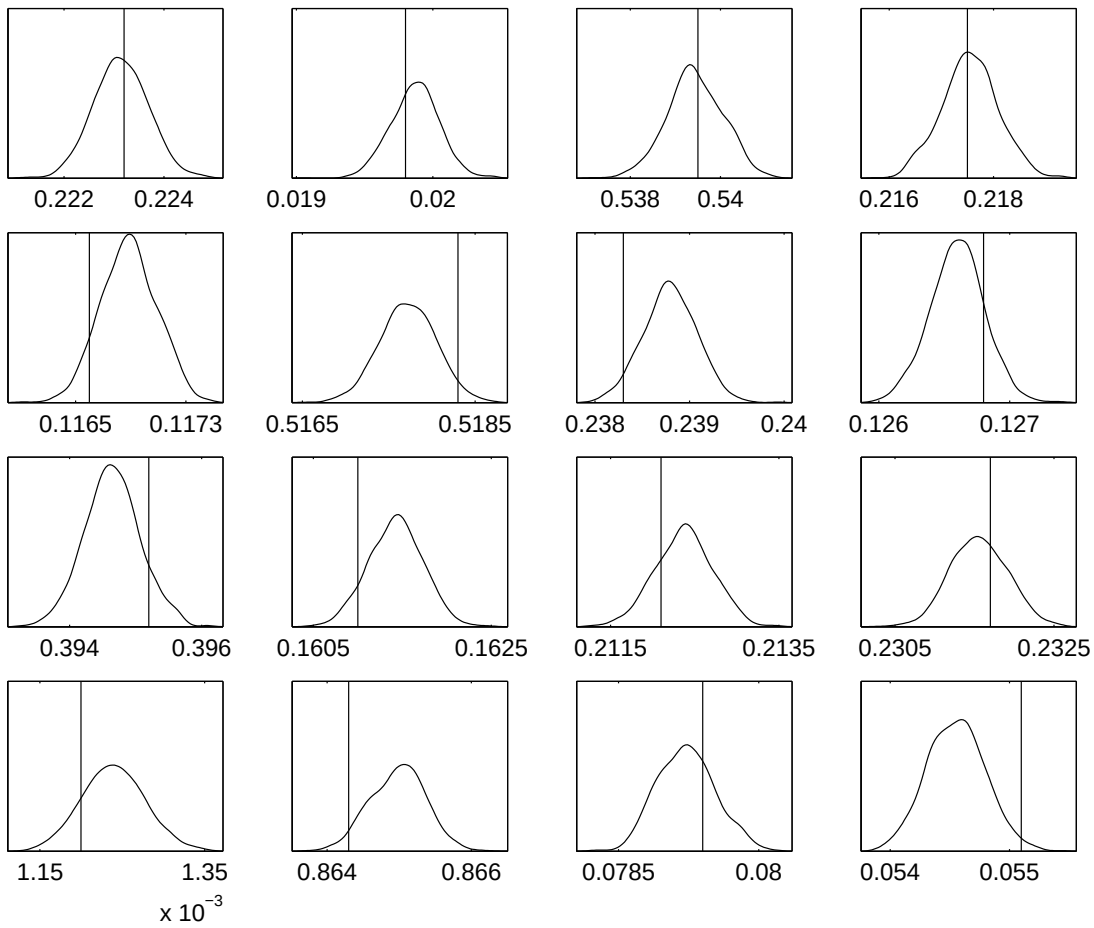


Figure 2.7: The posterior distribution of the transition probabilities for one of the transition matrices. 1PFS generated these densities. The vertical line marks the true value used to simulate the dataset.

Figure 2.8.

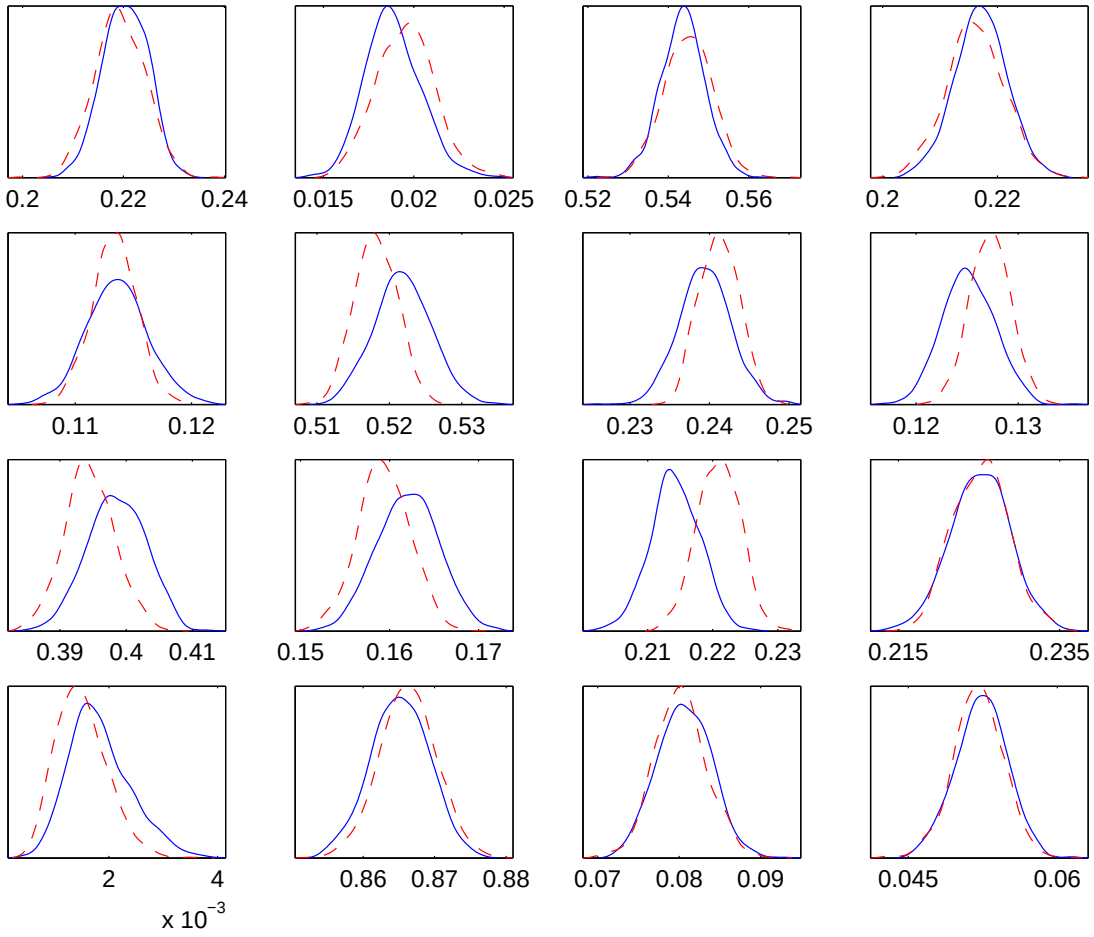


Figure 2.8: The posterior distribution of the transition probabilities for one of the transition matrices for a smaller subset of the whole dataset (the first 10,000 observations). The posterior density found via MCMC is represented by the blue solid line and that of the particle filter by the red dashed line.

Note that increasing M does not change the number of data accesses for 1PFS while each additional draw represents yet another scan for the standard implementation.

2.6 Discussion

MCMC has established itself as a standard tool for the statistical analysis of complex models. Data mining research, by contrast, rarely features MCMC. Indeed, for data mining applications involving massive datasets, computational barriers essentially preclude routine use of MCMC. MCMC is however an indispensable tool for Bayesian

analysis, especially in those applications where the inferential targets are more complex than posterior means or modes. As such we contend that extension of MCMC methods to larger datasets is an important research challenge. We present one particular line of attack. Working with samples from massive datasets represents an alternative strategy for large-scale Bayesian data analysis and may be viable for some applications. In high dimensional applications, however, throwing data away may be too costly.

Indeed, high dimensionality is well known to be the bane of all kernel density based approximation methods. However, it should be pointed out here that our proposed scheme utilizes kernel smoothing (and not density estimation per se) and thus should be applicable even in medium dimension problems (Liu and West 2001, have examples where they apply kernel smoothing in dimension around 30)[2].

By reducing the number of data accesses by a huge amount (one-pass for all but a small fraction of the data), MCMC becomes viable for a large class of models useful in data mining. We note that 1PFS can bypass the exhaustive analysis of an initial portion of the training data by sampling initial particles from the prior distribution of the parameters. While this doesn't affect any of the analysis, we have seen in practice, that it is often a good idea to start the particle filter with a reasonable set of particles. The sequential nature of the algorithm also allows the analyst to stop when uncertainty in the parameters of interests has dropped below a required tolerance limit. Parallelization of the algorithm is straightforward. Each processor manages a small set of the weighted draws from the posterior and is responsible for updating their weights and computing the refresh step.

References

J. Aitchison and I. J. Lauder. Kernel density estimation for compositional data. *Applied Statistics*, 34(2):129 – 137, 1985.

I. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White. Visualization of navigation patterns on a web site using model-based clustering. Technical report, Microsoft Research, 2000. Technical Report MSR-TR-00-18.

N. Chopin. A sequential particle filter method for static models. *Biometrika*, 89(3):539 – 552, 2002a.

[2]Software for both examples is available at: `http://paul.rutgers.edu/~suhrid/code/1PFS`.

N. Chopin. Central limit theorem for sequential monte carlo methods and its applications to bayesian inference. Technical report, CREST, 2002b. URL `http://www.crest.fr/doctravail/document/2002-44.pdf`.

A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, New York, NY, 2001.

W. DuMouchel. Bayesian data mining in large frequency tables, with an application to the fda spontaneous reporting system (with discussion). *The American Statistician*, 53(3):177 – 190, 1999.

M. Figueiredo. Adaptive sparseness using jeffreys prior. In S. Becker T. G. Dietterich and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems, (NIPS 14)*, Vancouver, Canada, 2001. MIT Press.

N. Friedman, I. Nachman, and D. Peer. Learning bayesian network structures from massive datasets: The sparse candidate algorithm.

A. Genkin, D. D. Lewis, and D. Madigan. Bayesian logistic regression for text categorization. 2004. In preparation.

J. Geweke. Bayesian inference in econometric models using monte carlo integration. *Econometrica*, 24:1317 – 1399, 1989.

W. Gilks and C. Berzuini. Following a moving target - monte carlo inference for dynamic bayesian models. *Journal of the Royal Statistical Society B*, 63(1):127 – 146, 2001.

A. Kong, J. Liu, and W. Wong. Sequential imputation and bayesian missing data problems. *Journal of the American Statistical Association*, 89:278 – 288, 1994.

J. Liu and M. West. *Combined parameter and state estimation in simulation-based filtering*, pages 197 – 224. Springer-Verlag, New York, NY, 2001.

P. Del Moral and A. Guionnet. A central limit theorem for nonlinear filtering using interacting particle systems. *Annals of Applied Probability*, 9:275 – 297, 1999.

C. Posse. Hierarchical model-based clustering for large datasets. *Journal of Computational and Graphical Statistics*, 10(3):464 – 486, 2001.

M. Ramoni, P. Sebastiani, and P. Cohen. Bayesian clustering by dynamics. *Machine Learning*, 47(1):91 – 121.

G. Ridgeway. Finite discrete markov process clustering. Technical report, Microsoft Research, 1997. Technical Report MSR-TR-97-24.

G. Ridgeway and D. Madigan. A sequential monte carlo method for bayesian analysis of massive datasets. *Journal of Knowledge Discovery and Data Mining*, 7:301 – 319, 2002.

G. Ronning. Maximum likelihood estimation of dirichlet distributions. *Journal of Statistical Computation and Simulation*, 32(4):215 – 221, 1989.

B. W. Silverman. *Density Estimation for Statistics and Data Analysis. Monographs on Statistics and Applied Probability*. Chapman Hall, New York, NY, 1986.

P. Stavropoulos and D. M. Titterington. *Improved particle filters and smoothing.* Springer-Verlag, New York, NY, 2001.

R. Tibshirani. Regression selection and shrinkage via the lasso. *Journal of the Royal Statistical Society, Series B*, 57:267 – 288, 1995.

Chapter 3

Sparse linear classifiers for Large d[1]

3.1 Introduction

Chapter 2 considered problems of massive data, but limited dimension ($d < 30$). In this chapter, we instead consider the problem of learning high-dimensional sparse linear classifiers from large numbers of training examples. A number of different applications from finance, text mining, and bioinformatics motivate this work. We concern ourselves specifically with binary classification and consider L_1-regularized logistic and probit regression models. Such models have provided excellent predictive accuracy in many applications (see, for example, Genkin et al., 2003; Figueiredo and Jain, 2001; Shevade and Keerthi, 2003) and attack overfitting and variable selection in a unified manner. L_1-regularization and a maximum *a posteriori* (MAP) Bayesian analysis with so-called Laplacian priors yield identical results (Tibshirani, 1996) and in order to streamline our presentation, we adopt the Bayesian approach. Many training algorithms now exist for L_1-logistic regression that can handle high-dimensional input vectors (Hastie et al., 2004; Shevade and Keerthi, 2003). However, these algorithms generally begin with a "load data into memory" step that precludes applications with large numbers of training examples. More precisely, consider a training dataset that comprises t examples each of dimension d. Due to matrix multiplications on $t \times t$ or $d \times d$ matrices, typical computational time requirements are $O(t^3 + d^3)$, with memory requirements that are $O(td + d^2)$. In our target applications, both t and d can exceed 10^6 so standard algorithms become impractical.

We present two basic algorithms for learning L_1-logistic and/or probit regression

[1]The basic content in this chapter has appeared in "Algorithms for Sparse Linear Classifiers in the Massive Data Setting", Balakrishnan S. and Madigan D., submitted manuscript, 2006.

models. Both operate in the data streaming model, by which we mean that they scan the data sequentially, and never require storing processed observations. The first algorithm we present is an online algorithm which sequentially processes each observation only once. This algorithm is provably non-divergent and uses in the worst case $O(d^2)$ time and $O(d^2)$ space to assimilate each new training example (note that both costs are constant with respect to the number of observations, t). Further, if the input data is sparse, the practical computational cost is better represented by $O(f^2 + md)$ where $f, m \ll d$ (we define the quantities f and m later).

For t constant, that is, for fixed but massive datasets, we also present a second algorithm that allows practitioners to trade-off computational time for improved accuracy. This multi-pass algorithm (the MP algorithm) also processes data sequentially but makes a small constant number of extra passes over the data set. The algorithm's computational cost is a thus constant factor higher and memory costs are essentially the same as the online algorithm. Finally, we propose the RMMP (Reduced Memory MP) algorithm that has significantly lower worst case memory costs, $O(d + k^2)$ (where $k \ll d$) and the same computational costs as the MP algorithm (thus both computational and memory costs are essentially linear in t and d). We will comment on the similarities and differences of our technique to other learning algorithms, in particular other online algorithms, in the following sections.

3.2 Background and Notation

As with the rest of this thesis, in this chapter, we will continue to concern ourselves with the task of binary classification, with class labels $y \in \{0, 1\}$. The training data comprise t labeled training examples, i.e., $D_t = \{(\mathbf{x}_i, y_i)\}_{i=1}^t$, with input vectors $\mathbf{x}_i = [x_{i_1}, \ldots, x_{i_d}]^T$ in $\mathbb{R}^d$ and corresponding labels y_i, $i = 1, \ldots, t$. We consider probabilistic classifiers of the form:

$$p(y = 1|\mathbf{x}) = \Phi(\boldsymbol{\beta}^T \mathbf{x})$$

where $\boldsymbol{\beta} \in \mathbb{R}^d$ is a vector of regression parameters and $\Phi(\cdot)$ is a link function. We restrict our analytical results to the two most commonly used link functions, the probit

$\Phi(z) = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx$ and logistic $\Phi(z) = \frac{e^z}{1+e^z}$ link functions.

The machine learning problem is thus to estimate the parameters $\boldsymbol{\beta}$, in the light of the training data D_t. We tailor our results towards high input dimension, that is, large d, and large numbers of training vectors, large t. Viewing the learning problem as one of Bayesian inference, we seek to compute the posterior distribution of the parameters $\boldsymbol{\beta}$ conditioned on a labeled training dataset D_t, given a prior distribution on the parameters $\boldsymbol{\beta}$:

$$p(\boldsymbol{\beta}|D_t) \propto \left(\prod_{i=1}^{t} p(y_i|\boldsymbol{\beta}) \right) p(\boldsymbol{\beta}). \tag{3.1}$$

The quantity on the left hand side of (3.1) is the required posterior distribution of $\boldsymbol{\beta}$ given the dataset D_t, while the second term on the right hand side is the prior distribution on $\boldsymbol{\beta}$, which we will specify momentarily. The first term on the right hand side is the likelihood:

$$\prod_{i=1}^{t} p(y_i|\boldsymbol{\beta}) = \prod_{i=1}^{t} \left(y_i \Phi(\boldsymbol{\beta}^T \mathbf{x}_i) + (1 - y_i)(1 - \Phi(\boldsymbol{\beta}^T \mathbf{x}_i)) \right). \tag{3.2}$$

Finding the MAP $\boldsymbol{\beta}$ leads to the optimization problem we wish to solve (now on the log scale):

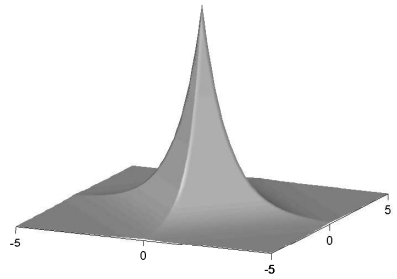
$$\begin{aligned} & \max_{\boldsymbol{\beta}} \left(\log p(\boldsymbol{\beta}|D_t) \right) \\ \equiv \quad & \max_{\boldsymbol{\beta}} \left(\sum_{i=1}^{t} \log \left(y_i \Phi(\boldsymbol{\beta}^T \mathbf{x}_i) + (1 - y_i)(1 - \Phi(\boldsymbol{\beta}^T \mathbf{x}_i)) \right) - \log p(\boldsymbol{\beta}) \right). \end{aligned}$$

The prior distribution $p(\boldsymbol{\beta})$ we pick for the parameters is the LASSO prior (Tibshirani, 1996), a product of independent Laplacian or double-exponential prior distributions on each component β_j:

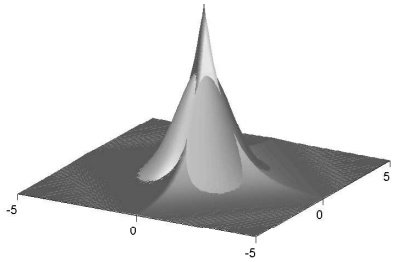
$$p(\beta_j|\gamma) = \frac{\gamma}{2} e^{-\gamma|\beta_j|}, \gamma > 0, j = 1, \ldots, d.$$

A prior of this form places high probability mass near zero and along individual component axes (Note: this is the same prior used in Chapter 2's regression example). It

also has heavier tails than a Gaussian distribution—see Figure 3.1 for plots of the 2-dimensional distributions. It thus favors locations in parameter space with component



(a)



(b)

Figure 3.1: (a) A standard Laplacian distribution, $\gamma = 1$ (b) A superposition of standard (zero mean, unit variance) Gaussian distribution, and the Laplacian distribution clearly showing both the higher probability mass the Laplacian assigns along the axes and at zero as well as its heavier tails.

magnitudes either exactly zero, and hence pruned from our predictive model, or shrunk towards zero. With this prior distribution, (3.3) presents a convex optimization problem and yields the same solutions as the LASSO (Tibshirani, 1996) and Basis Pursuit (Chen et al., 1999):

$$\begin{aligned} & \max_{\boldsymbol{\beta}} \left(\log p(\boldsymbol{\beta}|D_t)\right) \\ \equiv \quad & \max_{\boldsymbol{\beta}} \left(\sum_{i=1}^{t} \log \left(y_i \Phi(\boldsymbol{\beta}^T \mathbf{x}_i) + (1-y_i)(1-\Phi(\boldsymbol{\beta}^T \mathbf{x}_i)\right) - \gamma \|\boldsymbol{\beta}\|_1 \right). \end{aligned} \tag{3.3}$$

The parameter γ in the above problem controls the amount of regularization. Figure

3.2 shows a 2-dimensional visualization of how the objective function of the optimization problem changes as γ is varied. The choice of the regularization parameter is an important but separate question in itself (Efron et al., 2004; Hastie et al., 2004). While methods such as cross validation can be used to pick its value, we do not address such issues in this work, and we simply assume γ is some fixed, user-specified constant.

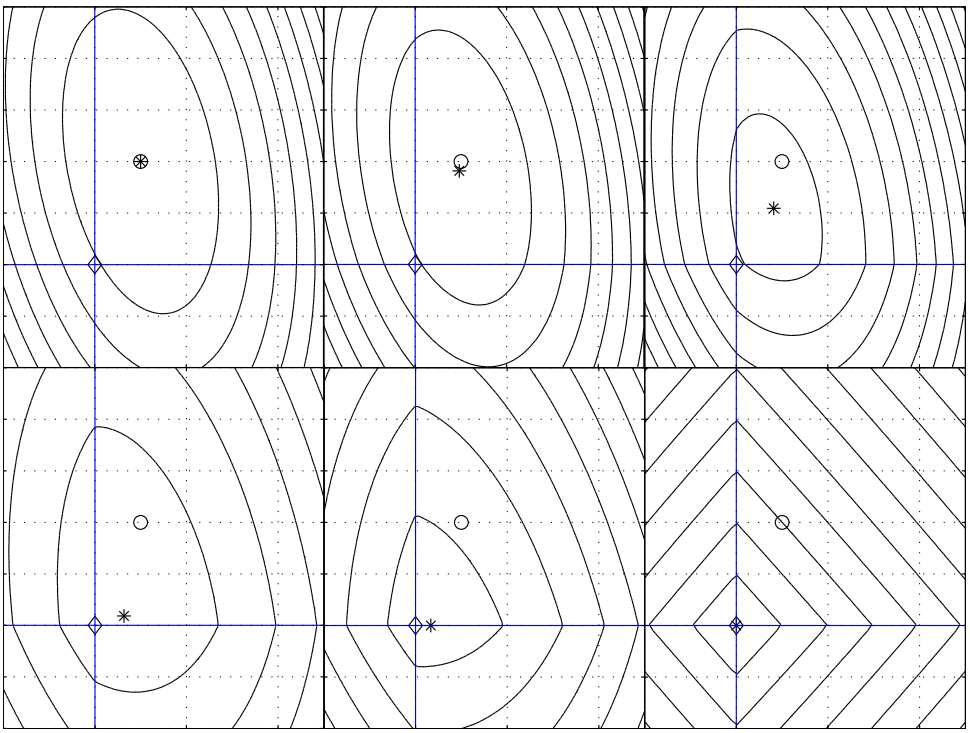


Figure 3.2: L_1-regularization in two dimensions (i.e., $d = 2$). The axes are the solid lines, the horizontal axis representing β_1 and the vertical axis representing β_2. The diamond represents the origin and the open circle represents the (non-regularized) maximum likelihood solution. The figure shows contours of the function in (3.3), the objective function, for increasing amounts of regularization (right to left and then top to bottom). The star shows the MAP location. The top row, left figure, shows negligible regularization; the MAP and maximum likelihood estimates coincide and the contours show no L_1-induced discontinuities. The top row, right figure, shows noticeable L_1 effects and the MAP and maximum likelihood solutions differ. The bottom row, middle panel shows enough L_1-regularization to set β_2 to zero (i.e., variable selection has occurred). The bottom row, right panel, shows extreme regularization, where both β_1 and β_2 are zero.

To the best of our knowledge, all existing algorithms solve the above convex optimization problem in the batch setting, i.e., by storing the dataset D_t in memory and

iterating over it (Fu, 1998; Osborne et al., 2000; Zhang and Oles, 2001; Zhang, 2002; Shevade and Keerthi, 2003; Genkin et al., 2003). Consequently, these algorithms cannot be used in the massive data/online scenario, where memory costs dependent on t must be avoided. The approach we present now attempts to overcome this limitation and thereby provide algorithms for training sparse linear classifiers without loading the entire dataset into memory.

3.3 Approximating the likelihood for online learning

The Bayesian paradigm supports online learning in a natural fashion; starting from the prior, the first training example produces a posterior distribution incorporating the evidence from the first example. This then becomes the prior distribution awaiting the arrival of the second example, and so on. In practice, however, except in those cases where the posterior distribution has the same mathematical form as the prior distribution, some form of approximation is required to carry out the sequential updating.

We want to avoid algorithms that begin with a "load data into memory" step and also avoid memory costs that increase with increasing amounts of data. In other words, we want memory costs independent of t, or be allowed to "forget" examples after processing them. We achieve this by maintaining the sufficient statistics of a standard quadratic approximation in $\boldsymbol{\beta}$ to the log-likelihood of the parameters for each observation.

We approximate the expression in (3.2) (on the log scale) as:

$$\begin{aligned}\sum_{i=1}^{t} \log(p(y_i|\boldsymbol{\beta})) &= \sum_{i=1}^{t} \log\left(y_i \Phi(\boldsymbol{\beta}^T \mathbf{x}_i) + (1-y_i)(1-\Phi(\boldsymbol{\beta}^T \mathbf{x}_i)\right) \\ &\approx \sum_{i=1}^{t} \left(a_i(\boldsymbol{\beta}^T \mathbf{x}_i)^2 + b_i(\boldsymbol{\beta}^T \mathbf{x}_i) + c_i\right),\end{aligned}$$

where $a_i(\boldsymbol{\beta}^T \mathbf{x}_i)^2 + b_i(\boldsymbol{\beta}^T \mathbf{x}_i) + c_i$ approximates $\log \Phi(\boldsymbol{\beta}^T \mathbf{x}_i)$ when $y_i = 1$ and approximates $\log(1 - \Phi(\boldsymbol{\beta}^T \mathbf{x}_i))$ when $y_i = 0$, $i = 1, \ldots, t$. In either case the approximation uses a simple Taylor expansion around $\boldsymbol{\beta}_{i-1}^T \mathbf{x}_i$, where $\boldsymbol{\beta}_{i-1}$ estimates the posterior mode given the first $i-1$ examples, D_{i-1} (Appendix 3.A provides expressions for a_i, b_i for

the probit and logistic link functions). We then have:

$$\begin{aligned}\sum_{i=1}^{t}\log(p(y_i|\boldsymbol{\beta})) &\approx \sum_{i=1}^{t}\left(a_i(\boldsymbol{\beta}^T\mathbf{x}_i)^2 + b_i(\boldsymbol{\beta}^T\mathbf{x}_i) + c_i\right)\\ &= \sum_{i=1}^{t} a_i(\boldsymbol{\beta}^T\mathbf{x}_i)(\mathbf{x}_i^T\boldsymbol{\beta}) + \sum_{i=1}^{t} b_i(\boldsymbol{\beta}^T\mathbf{x}_i) + \sum_{i=1}^{t} c_i\\ &= \boldsymbol{\beta}^T\boldsymbol{\Psi}_t\boldsymbol{\beta} + \boldsymbol{\beta}^T\boldsymbol{\theta}_t + \sum_{i=1}^{t} c_i\end{aligned}$$

where:

$$\boldsymbol{\Psi}_t = \sum_{i=1}^{t} a_i\mathbf{x}_i\mathbf{x}_i^T, \text{ and } \boldsymbol{\theta}_t = \sum_{i=1}^{t} b_i\mathbf{x}_i.$$

We now substitute this approximation of the log-likelihood function into equation (3.3) to obtain the modified (approximate) optimization problem:

$$\max_{\boldsymbol{\beta}}\left(\log p(\boldsymbol{\beta}|D_t)\right) \approx \max_{\boldsymbol{\beta}}\left(\boldsymbol{\beta}^T\boldsymbol{\Psi}_t\boldsymbol{\beta} + \boldsymbol{\beta}^T\boldsymbol{\theta}_t - \gamma\|\boldsymbol{\beta}\|_1\right). \tag{3.4}$$

Note that we can ignore the term involving the c_i's, as it is not a function of $\boldsymbol{\beta}$. Further, the fixed size $d \times d$ matrix $\boldsymbol{\Psi}$ and the $d \times 1$ vector $\boldsymbol{\theta}$ can be updated in an online fashion as data accumulate:

$$\boldsymbol{\Psi}_{t+1} = \boldsymbol{\Psi}_t + a_{t+1}\mathbf{x}_{t+1}\mathbf{x}_{t+1}^T, \text{ and } \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + b_{t+1}\mathbf{x}_{t+1}. \tag{3.5}$$

The size of the optimization problem in (3.4) doesn't depend on t, the size of the dataset seen so far. Thus, solving a fixed (with respect to t) size optimization problem allows one to sequentially process labeled data items and march through the dataset. In data streaming terminology, the matrix $\boldsymbol{\Psi}$ and the vector $\boldsymbol{\theta}$ provide a constant size sketch or summary of the labeled observations seen so far.

A number of questions now present themselves: how good is this approximation? How do we solve the approximate optimization problem efficiently? How does this approach differ from other likelihood approximation schemes (some which are also quadratic)? Also, the scheme as set up requires $O(d^2)$ memory in the worst case. Since we would like to use this approach for high dimensional datasets, can we reduce

the memory requirements?

The remainder of this chapter addresses these and other questions. First, we consider how to efficiently obtain the MAP solution of (3.4), the approximate optimization problem.

3.3.1 The modified Shooting algorithm

Recall that we need to find $\boldsymbol{\beta}$ that solves:

$$\max_{\boldsymbol{\beta}} \left(\boldsymbol{\beta}^T \boldsymbol{\Psi} \boldsymbol{\beta} + \boldsymbol{\beta}^T \boldsymbol{\theta} - \gamma \|\boldsymbol{\beta}\|_1 \right). \tag{3.6}$$

In the above equation and following discussion, we drop the subscript t from $\boldsymbol{\Psi}, \boldsymbol{\theta}$ for notational convenience. This is a convex optimization problem and a number of efficient techniques exist to solve it. Newton's method and other Hessian-based algorithms may be prohibitively expensive as they need $O(d^3)$ computational time in order to construct the Hessian/invert $d \times d$ matrices. Other authors have described good results on the arguably tougher (non-approximate) optimization problem for logistic regression (essentially the terms in Equation 3.3, but with L_2 regularization of $\boldsymbol{\beta}$) with techniques such as fixed memory BFGS (Minka, 2000), modified conjugate gradient (Moore and Komarek, 2004) and cyclic coordinate descent (Zhang and Oles, 2001; Genkin et al., 2003).

Here, we employ instead a slight modification of the Shooting algorithm (Fu, 1998), see Algorithm 1. Shooting is essentially a coordinate-wise gradient ascent algorithm, explicitly tailored for convex L_1-constrained problems. The vector $\boldsymbol{\Omega}$ in the algorithm is defined as $\boldsymbol{\Omega} = 2\boldsymbol{\Psi}'\boldsymbol{\beta} + \boldsymbol{\theta}$, where $\boldsymbol{\Psi}'$ is the matrix $\boldsymbol{\Psi}$ with its diagonal entries set to zero (see Appendix 3.B for details). This vector is related to the gradient of the differentiable part of the objective function and consequently can be used for optimality checking. Minor variants of this algorithm have been independently proposed by Shevade and Keerthi (2003) and Krishnapuram et al. (2005). Although Fu originally derived the algorithm by taking the limit of a modified Newton-Raphson method, it can also be obtained by a subgradient analysis of the system, subgradients being necessary due to

the non-differentiability that the L_1 constraints on $\boldsymbol{\beta}$ result in, see Appendix 3.B for the derivation.

Algorithm 1: The modified Shooting algorithm.

Data: $\boldsymbol{\Psi}, \boldsymbol{\theta}, \boldsymbol{\beta}_0, \gamma$.
$\boldsymbol{\beta}_0$ is initial $\boldsymbol{\beta}$ vector.
Ω_j refers to the j'th component of $\boldsymbol{\Omega}$.
Ψ_{jj} refers to the (j,j)'th element of matrix $\boldsymbol{\Psi}$.
Result: $\boldsymbol{\beta}$ satisfying (3.6).
while *not converged* **do**
  for $j \leftarrow 1$ **to** d **do**

$$\beta_j = \begin{cases} 0, & \text{if } |\Omega_j| \leq \gamma \\ \frac{\gamma - \Omega_j}{2\Psi_{jj}}, & \text{if } \Omega_j > \gamma \\ \frac{-\gamma - \Omega_j}{2\Psi_{jj}}, & \text{if } \Omega_j < -\gamma \end{cases}$$

    Update $\boldsymbol{\Omega}$.
  end
end

While one can think of numerous stopping criteria for the algorithm, we stop when successive iterates are sufficiently close to each other (relatively, and with respect to the L_2 norm). More precisely, we declare convergence whenever $\|\boldsymbol{\beta}_i - \boldsymbol{\beta}_{i-1}\|_2 / \|\boldsymbol{\beta}_{i-1}\|_2$ is less than some user specified tolerance.

In the worst case, each iteration of Shooting requires $O(d^2)$ computational time. However, for reasonable amounts of regularization, where the final set of non-zero $\boldsymbol{\beta}$ values is small, the time requirements are much smaller. Indeed, the practical computational cost is perhaps better reflected by bounds in terms of the sparsity of MAP $\boldsymbol{\beta}$. Let m denote the maximum number of non-zero components of $\boldsymbol{\beta}$ along the solution path to MAP $\boldsymbol{\beta}$ (hence $m \leq d$). Implemented carefully, Shooting requires $O(md)$ time per iteration (see Appendix 3.B for details).

While coordinate-wise approaches are commonly regarded as slow in the literature (for example, Minka, 2001a), for sparse classifiers, they are much faster (see for example, Shevade and Keerthi, 2003). In our experiments, the Shooting algorithm has proven to be practical even for d in the hundreds of thousands.

3.4 Related work

Approximating the log-likelihood function by a quadratic polynomial is a standard technique in Bayesian learning applications; see for example Laplace approximation (Kass and Raftery, 1995; MacKay, 1995), Assumed Density Filtering (ADF)/Expectation Propagation (EP) (Minka, 2001b), some variational approximation methods such as Jaakkola and Jordan (2000) and in Bayesian online learning (Opper, 1996). Our approach is closest in spirit to the online Bayesian method presented in Opper (1996) but is closer in the details of the approximation to ADF/EP as described in Minka (2001b). We briefly outline these similarities at a high level here; interested readers should refer to the original articles for details.

As stated in the previous section, although the Bayesian paradigm admits sequential updating of the posterior distribution (online learning) in a natural way, some form of approximation is almost always necessary for practical applications. Approximating the posterior distribution at every stage by a multivariate Normal (Gaussian) distribution seems a natural first step backed by asymptotic Bayesian central limit results that imply this approximation will get better and better with the addition of data (Bernardo and Smith, 1994).

The version of Assumed Density Filtering closest to our approach is described in Minka (2001b). The posterior distribution is assumed to be multivariate Normal and observations are processed sequentially. The posterior distribution is updated using the exact likelihood. This exact update typically results in the posterior distribution becoming some non-multivariate Normal distribution. This distribution is then approximated by a new multivariate Normal distribution (the approximation being based on minimizing the KL-divergence between the two distributions, achieved by matching moments) before processing the next observation, and so on—see Figure 3.3.

As Minka points out (Minka, 2001b), the exact update followed by approximation/projection described above can also be thought of in a different but equivalent way. Since when using ADF, the prior and posterior distributions are both multivariate Normal, we can think of the observation likelihood as first being approximated by

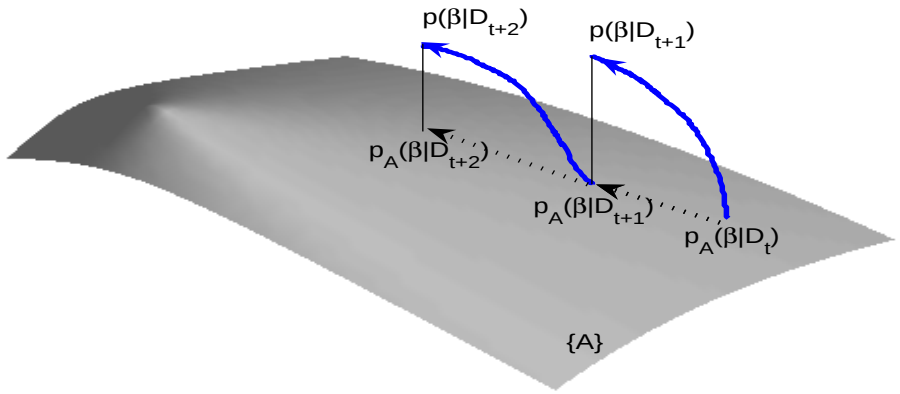


Figure 3.3: Schematic elucidating quadratic approximation schemes in literature (modified version of a figure in Solla and Winther, 1998). The surface pictorially represents the set of multivariate Normal distributions, $\{A\}$. Starting with a distribution in this family, $p_A(\boldsymbol{\beta}|D_t)$ (the subscript A denoting the distribution to be from $\{A\}$), incorporating the likelihood of the $t+1$'st observation exactly (shown by the heavy solid arrow) leads to a posterior distribution that is outside $\{A\}$, shown in the Figure as $p(\boldsymbol{\beta}|D_{t+1})$. This distribution is then projected back to $\{A\}$, giving $p_A(\boldsymbol{\beta}|D_{t+1})$, and this process continues. As Minka points out (Minka, 2001b), this is entirely equivalent to an exact update of $p_A(\boldsymbol{\beta}|D_t)$ by an appropriate quadratic approximation to the likelihood (the dashed arrow).

a multivariate Normal and then combined exactly with the prior to yield a posterior distribution in the same family of distributions—see Figure 3.3. The online Bayesian algorithm presented in Opper (1996) fits into this framework. In Opper's algorithm, the prior and posterior distribution are also assumed to be multivariate Normal. However, his Gaussian approximation to the likelihood of each observation is constructed differently and derives from mean-field theory considerations.

Similarly, Jaakkola and Jordan (2000), present a variational approximation scheme for logistic regression models that uses a different quadratic approximation to the log-likelihood. They explicitly seek to bound the true posterior distribution and their quadratic approximation includes an additional variational parameter.

Our approach differs from the above methods in the following ways: a) Our results hold for the Laplacian (sparsity favoring) prior. This is an important difference, because sparsity may entail better generalization performance. b) We explicitly design algorithms considering the high dimensionality of the input data; all of the alternative approaches do not scale to such datasets because their update steps involve inverting matrices of size $O(d^2)$. c) We propose an algorithm that applies to the online scenario (Opper's algorithm does as well). d) We are only concerned with the MAP estimate of $\boldsymbol{\beta}$ - all of the above approaches explicitly maintain both the mean and covariance matrix of the approximate posterior distribution (and thus have memory costs $O(d^2)$). e) Our algorithms exploit any inherent sparsity the input data may have.

3.5 An Online algorithm

The quadratic approximation and the Shooting algorithm lead straightforwardly to an online algorithm. After initializing the parameters $\boldsymbol{\Psi}_0, \boldsymbol{\theta}_0, \boldsymbol{\beta}_0$, process the dataset one observation at a time. Calculate the quadratic Taylor approximation to each observation's log-likelihood at the current estimate of the posterior mode, $\boldsymbol{\beta}_{i-1}$, thus finding parameters a_i, b_i. Use these parameters and the observation to update the sketches, $\boldsymbol{\Psi}, \boldsymbol{\theta}$. Now run the modified Shooting algorithm to update the posterior mode, $\boldsymbol{\beta}_i$ and repeat for the next labelled observation—see Algorithm 2.

Algorithm 2: The Online algorithm.

Data: D_t, γ.
Result: Sequentially outputs $\boldsymbol{\beta}_i$, the MAP estimate of $\boldsymbol{\beta}$ that solves (3.4).
Initialize $\boldsymbol{\beta}_0 = \theta_0 = \mathbf{0}$, $\boldsymbol{\Psi}_0 = \mathbf{0}$, $i = 1$.
while $i < t$ **do**
  Get i'th observation $(\mathbf{x}_i, y_i)$.
  Obtain quadratic approximation to term likelihood at $\boldsymbol{\beta}_{i-1}$, i.e., obtain a_i, b_i.
  $\boldsymbol{\Psi}_i \leftarrow \boldsymbol{\Psi}_{i-1} + a_i \mathbf{x}_i \mathbf{x}_i^T$.
  $\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_{i-1} + b_i \mathbf{x}_i$.
  $\boldsymbol{\beta}_i \leftarrow$ modified Shooting$(\boldsymbol{\Psi}_i, \boldsymbol{\theta}_i, \boldsymbol{\beta}_{i-1}, \gamma)$
  $i \leftarrow i + 1$.
end

We show the performance of the online algorithm on a low dimensional simulated dataset in Figure 3.4 (the data generating mechanism is a logistic regression model with $d = 11$ and $t = 100,000$. For details see the Experiments section). As we process greater numbers of observations, the online estimates (the solid lines) improve i.e., get closer to the batch estimates (the dashed lines which we obtain using BBR, publicly available software for batch L_1 penalized logistic regression). See Figure 3.4, where different colors represent different components of MAP $\boldsymbol{\beta}_i$. Figure 3.5 shows individual plots of the online and batch estimates for four representative components of MAP $\boldsymbol{\beta}_i$ in blue. We also plot the absolute difference between the batch and online estimates in green on the same plot on the right (green) axis. As we expect, after the parameter estimates stabilize, this difference steadily tapers off with increasing amounts of data.

In the worst case, the online algorithm requires $O(d^2)$ space and $O(d^2)$ computational time to compute the MAP $\boldsymbol{\beta}$ for each new observation. Note however, that if the input data has sparsity, which is true of text data for example, the algorithm leverages this. Let the maximum number of non-zero components in any $\mathbf{x}$ be f and assume a constant number of iterations of the modified Shooting algorithm. In such case, the practical computational time requirement of the algorithm is $O(f^2 + md)$ per observation. Although the practical memory costs of the algorithm will likely be less than $O(d^2)$, exactly how much less depends heavily on the data, since $\boldsymbol{\Psi}$ (the part of the sketch dominating the memory requirements) is a weighted sum of outer products of the $\mathbf{x}_i$'s. It is quite possible that even very sparse data may result in the full $O(d^2)$

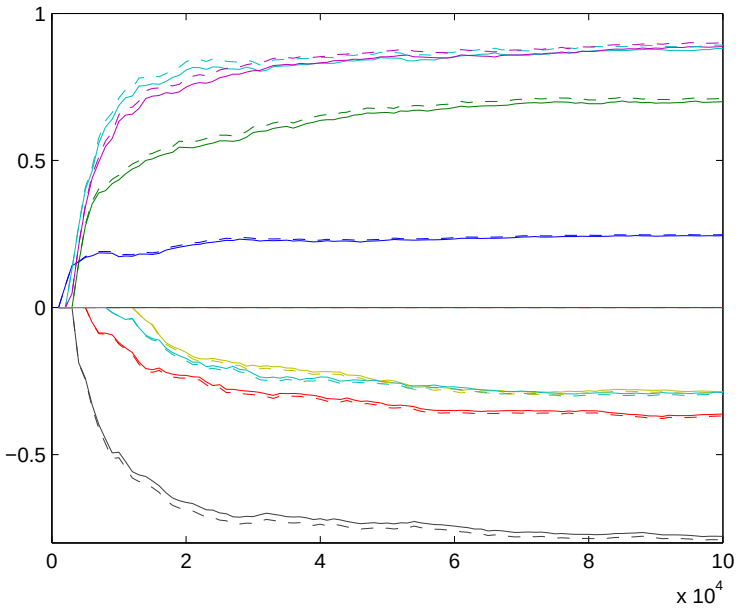


Figure 3.4: Performance of the online algorithm. Simulated dataset, $\gamma = 100$. The y-axis is the parameter value, the x-axis the number of observations processed, t.

memory requirement.

Here, we highlight the fact that the online algorithm is accurate and practical if the problem is of low to medium input dimension, but massive in terms of the number of observations. Appendix 3.C proves non-divergence of the algorithm in the infinite data limit.

3.5.1 Heuristics for improvement/Issues

While one can also obtain parameter estimates for fixed t using the online algorithm, multiple passes typically provide better estimates, albeit with increased computational cost. Denote by $\boldsymbol{\beta}_*$ the solution to the exact optimization problem (3.3) for some fixed t. Since the online algorithm typically initializes itself far from β_*, it is only after processing a sufficient number of examples that the online algorithm's term approximations will start being taken closer to $\boldsymbol{\beta}_*$. The update formulae, (3.5), reveal that for values of $i < t$, both $\boldsymbol{\Psi}_i$ and $\boldsymbol{\theta}_i$ are (comparatively) smaller in magnitude than their respective final values, $\boldsymbol{\Psi}_t, \boldsymbol{\theta}_t$. However, the amount of regularization remains relatively fixed at

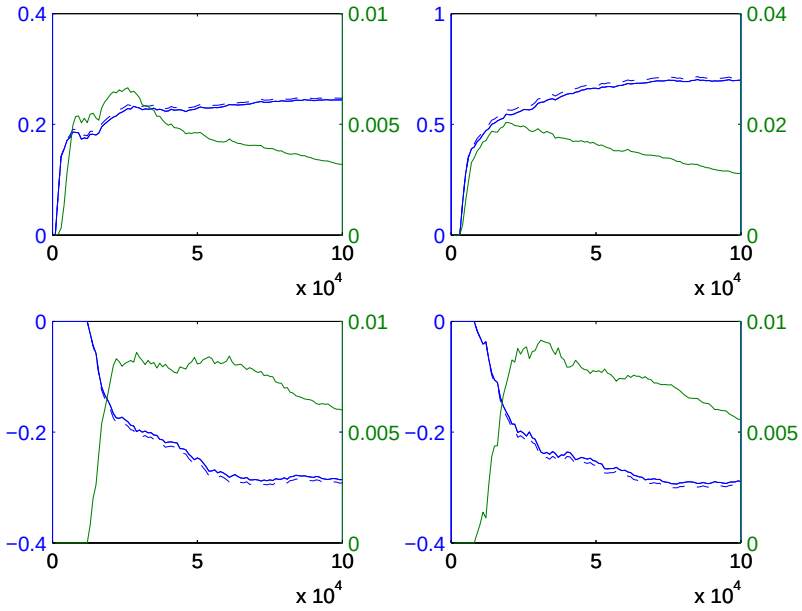


Figure 3.5: Slightly more detailed version of Figure 3.4. The panels show four representative parameters from that figure, also showing tapering L_1 loss between the online and batch algorithm estimates on the right axis (in green). Simulated dataset, $\gamma = 100$. Once again, the (left) y-axis is the parameter value and the x-axis the number of observations processed, t.

$\gamma\|\boldsymbol{\beta}\|_1$. Hence, if the online algorithm is initialized at $\boldsymbol{\beta}_0 = \mathbf{0}$, for any $i < t$, the output MAP estimate $\boldsymbol{\beta}_i$ will be more shrunk towards zero than $\boldsymbol{\beta}_*$. Figure 3.4 illustrates this for smaller values of t where the solid lines (approximate MAP estimates) are closer to zero than the dashed lines (exact batch estimates).

This suggests the following two heuristics to improve the quality of estimates from the online algorithm. The first is to increase the amount of regularization gradually as the algorithm processes observations sequentially (via a schedule, linearly say, $\propto t$ from zero initially to the specified value γ at the end of the dataset). Less regularization of the first few observations somewhat mitigates the effect of taking term approximations at shrunken parameter estimates.

The second heuristic is for the online algorithm to keep a block of observations in memory. The algorithm then uses the value of the parameter estimates after having seen/processed all the observations in a block to update the sketch for the whole block.

	$t = 2\text{x}10^4$		$t = 6\text{x}10^4$		$t = 10^5$	
$\boldsymbol{\beta}_{true}$	**Batch**	**Online**	**Batch**	**Online**	**Batch**	**Online**
0.259	0.244	0.242	0.248	0.247	0.254	0.253
0.761	0.700	0.690	0.743	0.739	0.740	0.737
-0.360	-0.360	-0.356	-0.401	-0.399	-0.394	-0.393
0.876	0.980	0.966	0.918	0.913	0.922	0.919
0.913	0.920	0.907	0.920	0.916	0.931	0.929
-0.302	-0.275	-0.270	-0.327	-0.324	-0.317	-0.315
-0.820	-0.826	-0.814	-0.806	-0.802	-0.819	-0.816
0	0	0	-0.010	-0.010	-0.005	-0.005
0	0.050	0.049	0	0	0.013	0.013
0	0.038	0.037	0.014	0.014	0.013	0.013
-0.319	-0.298	-0.294	-0.318	-0.316	-0.320	-0.319
L_1 Norm	0.066		0.025		0.016	

Table 3.1: Table with columns showing values of $\boldsymbol{\beta}_{true}$, and the MAP estimates of $\boldsymbol{\beta}$ obtained by the batch algorithm and the online algorithm, for increasing amounts of data on the simulated dataset. To aid assessing convergence of the online to the batch estimates, we show the value of the L_1 norm of the adjacent vectors (batch vs. online estimates) in the last row. For this example, $\gamma = 10$ (logistic link function).

Note that this will involve keeping track of the corresponding updates to the sketches for the block. In experiments not reported here, both of these heuristics improve the final online estimates somewhat.

One possibility for improving upon the $O(d^2)$ worst case computational requirement of the online algorithm is as follows. In the infinite data case, in order to obtain sparsity in parameter estimates, the amount of regularization must be allowed to increase as observations accumulate—an increasingly weighty likelihood term will inundate any fixed amount of regularization. In this setting (where we have the freedom to choose the amount of regularization), we can use exactly the same quadratic approximation machinery to pick the value of γ that minimizes the approximate one-step look ahead predictive error (the expressions for approximating ramp loss would be almost identical to those for the log-likelihood). The resulting scheme has the flavor of predictive automatic relevance determination as presented in Qi et al. (2004).

The worst case $O(d^2)$ memory requirement of the online algorithm, however, presents a greater challenge. In the next section we outline a multi-pass algorithm based on the same sequential quadratic approximation that improves the accuracy of estimates when applied to finite datasets and also uses less memory than the online algorithm.

3.6 A multi-pass algorithm

Continuing with the heuristic for the online algorithm, taking *all* term approximations at the final online algorithm MAP $\boldsymbol{\beta}_t$ value would certainly produce better estimates of $\boldsymbol{\Psi}_t, \boldsymbol{\theta}_t$ that would consequently result in a better estimate of $\boldsymbol{\beta}_*$.

Therefore, for fixed datasets where computational time restrictions still permit a few passes over the dataset, this suggests the following algorithm, which we will refer to as the MP (Multi-Pass) algorithm: Initialize $\boldsymbol{\beta}_0 = \theta_0 = \mathbf{0}$, $\boldsymbol{\Psi}_0 = \mathbf{0}$, $z = 1$. The quantity z will count the number of passes through the dataset. Compute $\boldsymbol{\Psi}_t, \boldsymbol{\theta}_t$ by the steps in Online Algorithm (Algorithm 2), except take *all* term approximations at the *fixed value* $\boldsymbol{\beta}_z$. Note that consequently there is no need for the shooting algorithm during the pass through the dataset. Once a pass through the dataset is complete, compute a revised estimate of $\boldsymbol{\beta}_*$ by running modified Shooting, i.e., set $\boldsymbol{\beta}_{z+1}$ =modified Shooting$(\boldsymbol{\Psi}_t, \boldsymbol{\theta}_t, \boldsymbol{\beta}_z, \gamma)$. Iteratively loop over the dataset, appropriately incrementing z.

Although not strictly comparable, the MP algorithm is very similar to Expectation Propagation (Minka, 2001), where passes through the dataset result in better term approximations, and hence a better final estimate of the MAP $p(\boldsymbol{\beta}|D_t)$. Like EP, the above scheme typically requires only a few passes through the dataset to converge. For a constant number of passes, the MP algorithm has the worst case computational time requirement of $O(td^2)$ to do an equivalent batch MAP $\boldsymbol{\beta}$ estimation. Once again, if the dataset is sparse, this cost is closer in practice to $O(tf^2 + md)$.

The worst case memory requirement of the MP algorithm is $O(d^2)$, which is fixed with respect to t (EP requires explicitly storing term approximations and thus has memory costs $O(t)$). The next subsection presents a modification of the MP algorithm that reduces this worst case memory requirement.

3.6.1 A reduced memory multi-pass algorithm

The key to reducing the memory requirements of the algorithm in the previous subsection is exploiting the sparsity of $\boldsymbol{\beta}_*$. Towards this end, consider the modified Shooting

algorithm upon convergence; say $\boldsymbol{\beta}_{MAP}$ is the sparse converged solution Shooting obtains with inputs $\boldsymbol{\Psi}, \boldsymbol{\theta}$ and γ. Now consider the smaller system obtained by only retaining those rows of the vectors, and also corresponding columns for matrices, for which the components of $\boldsymbol{\beta}_{MAP}$ are nonzero (denoted with a $\tilde{}$). The important observation is that the solution to the reduced size system $\tilde{\boldsymbol{\beta}}_{MAP}$, obtained using $\tilde{\boldsymbol{\Psi}}, \tilde{\boldsymbol{\theta}}$ and $\tilde{\boldsymbol{\Omega}}$, has exactly the same nonzero components as $\boldsymbol{\beta}_{MAP}$ obtained for the full system.

We use this fact to derive the RMMP (Reduced Memory Multi-Pass) algorithm, Algorithm 3. The core of the algorithm is the same as before. However, we now keep track of a much smaller matrix $\tilde{\boldsymbol{\Psi}}$, while also keeping track of the original vectors $\boldsymbol{\theta}$ and $\boldsymbol{\Omega}$. The update for $\boldsymbol{\theta}$ is unchanged and Appendix 3.B shows how to perform the update for $\boldsymbol{\Omega}$ in small space. The central idea is to use the optimality criteria for the Shooting algorithm to determine which components of $\boldsymbol{\beta}$ to keep track of (call this set S, the active set, which is fixed during every iteration). Specifically, we set $S = \{j : |\Omega_j| \geq \gamma\}$, i.e., the set of variables that are either nonzero and optimal or variables that violate optimality in every pass (the corresponding nonzero elements of the vectors/matrices are denoted by their previous symbols but with a $\tilde{}$ above them).

A desirable consequence of the setup is that no new approximation is introduced. The search for the optimal parameter values is slightly more involved though, now proceeding iteratively by first identifying candidate nonzero components of $\boldsymbol{\beta}_{MAP}$, and then refining the estimates for these components. We can employ the same stopping criteria as for modified Shooting algorithm.

Note that memory requirements are now $O(d + k^2)$, where k is the number of variables in the largest active set. However, we can be even more stringent and set k to be a user specified constant provided k is bigger than the final number of nonzero components of $\boldsymbol{\beta}_*$. Typically, setting k very close to this limit results in some loss of accuracy and the cost of a few more passes over the data for convergence. The worst case computational requirements for a constant number of passes, are still $O(td^2)$ to do an equivalent batch MAP $\boldsymbol{\beta}$ estimation and in a manner similar to before, practically better reflected by $O(t(k^2 + f^2) + kd)$.

Algorithm 3: The RMMP algorithm.

Data: fixed dataset D_t, γ.
Result: $\boldsymbol{\beta}_z$, the MAP estimate of $\boldsymbol{\beta}$ that solves (3.3).
Initialize $\boldsymbol{\beta}_0 = \mathbf{0}$, $S = \{\}, z = 1$.
while *not converged* **do**
  Set $\boldsymbol{\theta} = \mathbf{0}$, $\tilde{\boldsymbol{\Psi}} = \mathbf{0}$, $i = 1$.
  for $i = 1, 2, \ldots, t$ **do**
    Get i'th observation $(\mathbf{x}_i, y_i)$.
    Obtain quadratic approximation to term likelihood at $\boldsymbol{\beta}_{z-1}$, i.e., obtain a_i, b_i.
    $\tilde{\boldsymbol{\Psi}} \leftarrow \tilde{\boldsymbol{\Psi}} + a_i(\mathbf{x}_i\tilde{\mathbf{x}}_i^T)$.
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + b_i\mathbf{x}_i$.
    Update $\boldsymbol{\Omega}$.
  end
  $\boldsymbol{\beta}_z \leftarrow$ modified Shooting$(\tilde{\boldsymbol{\Psi}}, \tilde{\boldsymbol{\theta}}, \tilde{\boldsymbol{\beta}}_{z-1}, \gamma)$.
  Obtain new active set $S = \{j : |\Omega_j| \geq \gamma\}$.
  $z \leftarrow z + 1$.
end

We now draw attention to a few practical considerations about the RMMP algorithm. The first is that although we consider initializing the parameter vector to zero, $\boldsymbol{\beta}_0 = \mathbf{0}$, better guesses of $\boldsymbol{\beta}_0$ (guesses closer to the MAP $\boldsymbol{\beta}$) would likely result in less passes for convergence. Further, given we do initialize at zero, the first pass is completed very rapidly. This is because no outer products are computed, since the active set is initialized as the empty set. Indeed in this case the first pass is used simply to determine the size and components of the active set; the parameter estimates for the next iteration are still zero, $\boldsymbol{\beta}_1 = \mathbf{0}$. Typically, setting the reduced memory parameter k to be larger than this first active set size results in further RMMP iterations mimicking iterations of the MP algorithm. This is seen by observing two facts. One, for both algorithms, the only components that change in successive iterations are those in the active set (components that are either non-zero and optimal or not optimal). Two, in a typical search path for the MAP $\boldsymbol{\beta}$, the size of the active set decreases (and finally stabilizes) as the MAP $\boldsymbol{\beta}$ is honed in on. Both of these observations together imply that if we start the RMMP algorithm with enough memory allotted to look at all possibly relevant $\boldsymbol{\beta}$ components, we will follow the MP search path (as a motivating example, consider that setting $k = d$ results in the MP algorithm exactly).

Another consideration is a very useful practical advantage of the proposed algorithm: knowledge of $\mathbf{\Omega}$ implies the practitioner can confirm when convergence to $\boldsymbol{\beta}_*$ has/has not occurred. We also point out that in practice, for numerical stability, slightly expanding the active set seems to be a good heuristic. In our experiments that follow, we do so only if we have extra space (if $k > |S|$ for any iteration) in two ways: 1. We retain in the active set variables that were in the previous active set and, 2. we add to the active set components that are *close to* violating optimality (close in terms of a threshold, $\tau < 1$. This amounts to replacing the rule in Algorithm 3 with $S = \{j : |\Omega_j| \geq \tau\gamma\}$).

The next section describes results we obtained on some simulated as well as real examples using the proposed algorithms.

3.7 Experiments

We now present examples illustrating the application of the MP and RMMP algorithms to simulated datasets, where we control the data generating mechanism, and some real datasets. We make logistic regression comparisons to results obtained using BBR (Genkin et al., 2003). BBR is publicly available software for Bayesian binary logistic regression that handles the Laplacian prior. We make probit regression comparisons to results obtained using a batch EM algorithm for Laplacian prior based probit regression (a slightly modified version of the algorithm in Figueiredo and Jain, 2001). We generally do not present prediction accuracy results here as our goal is to obtain accurate, i.e., close to batch, parameter values. What we wish to accomplish with the experiments is demonstrate practical efficiency and applicability of the algorithms. In so doing and by obtaining essentially identical parameter estimates to batch algorithms, our predictive performance will mirror those of the batch algorithms. Several papers provide representative predictive performance results for L_1-regularized classifiers, for example, Genkin et al. (2003); Figueiredo and Jain (2001).

We carried out all the experiments on a standard Windows OS based 2Ghz processor machine with 1GB RAM. For all experiments we set the modified Shooting convergence tolerance to be 10^{-6}, and $\tau = 0.8$ (for experiments involving the RMMP algorithm).

We use the following datasets:

- Simulated datasets: d=11, t=10,000. The data generating mechanism is either a probit or logistic regression model with 11 known parameters, of which three are intentionally set as redundant variables (set with zero coefficients in the model). For the experiments with the online algorithm (Figure 3.4, Table 3.1), we used the same model parameters as above, but with $t = 100,000$ and only a logistic regression model.
- ModApte training dataset: $d = 21,989$, $t = 9,603$. This is a text dataset, the ModApte split of Reuters-21578 (Lewis, 2004). We examine one particular category, money-fx, to which we fit a logistic regression model.
- BIG-RCV dataset: $d = 288,062$, $t = 421,816$, a dataset constructed from the RCV1-v2 dataset (Lewis et al., 2004). It consists of the training portion of the LYRL2004 split plus 2 parts of the test data (the test data is made publicly available in $4 \approx 350$ MB parts)—see Figure 3.6. We also use just the training portion of RCV1-v2 in some experiments. RCV1-v2 training dataset : $d = 47,152$, $t = 23,149$ (the features in this dataset are a particular subset of the features in BIG-RCV). Our results are for a single topic "ECAT", whether or not a document is related to economics.

3.7.1 Results

The small dimensional simulated dataset highlights typical results we obtain with the MP algorithm (the RMMP algorithm is not of practical significance in this case). With very few additional passes over the dataset, denoted as before by the variable z, we obtain parameter estimates practically identical to those obtained by the batch algorithm—see Table 3.2. Each column in the table is an 11-dimensional vector which is the MAP $\boldsymbol{\beta}$ estimate of the parameter values. The table reports typical results we get for both link functions and over a wide range of γ values. As a guide to assessing convergence, the Tables show the L_1 norm of the difference between the batch algorithm estimates (EM or BBR as appropriate) and the MP (or RMMP, as appropriate) algorithm iterates.

We next examine the first real dataset, the training data for the ModApte split of Reuters-21578 (Lewis et al., 2004). This is a moderate dimensional ($d = 21989$ features)

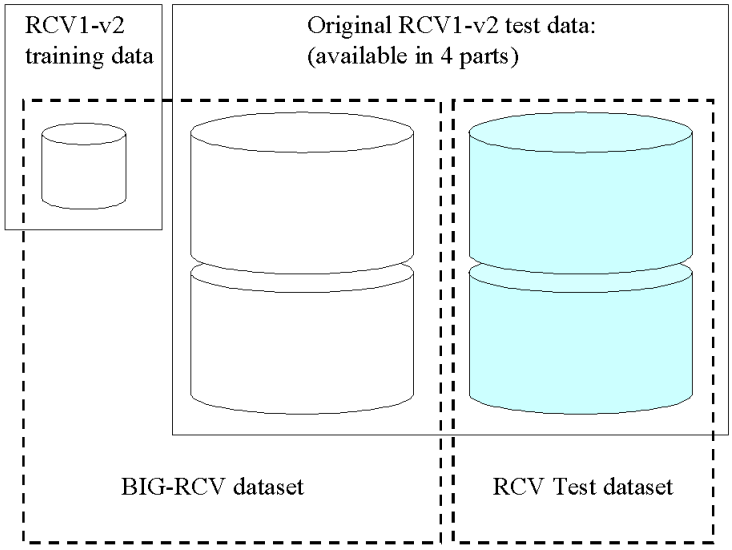


Figure 3.6: Schematic showing the construction of the various RCV1-v2 based datasets used in the experiments. The solid line bordered rectangles show the data as publicly available, the dashed-line bordered rectangles show the datasets we assembled. The shaded portion of the data is used only during testing.

yet manageable sized dataset with $t = 9603$ labelled observations. The features of this dataset are weighted term occurrences and it is quite sparse, as is typical for text data. The batch EM algorithm for probit regression is prohibitively expensive on this dataset as it involves inverting a very high dimensional matrix, but we can run BBR to obtain batch logistic regression results. Hence we focus our results on logistic regression for this dataset. We examine two reasonable settings for the regularization parameter, $\gamma = 10$ and $\gamma = 100$. For $\gamma = 10$, BBR returns 150 nonzero components and for $\gamma = 100$, the MAP $\boldsymbol{\beta}$ BBR returns has 31 non-zero components. Since the dataset is sparse, we are able to apply both the MP and RMMP algorithms.

For $\gamma = 100$, the MP algorithm converges in about $z = 6$ iterations to parameter values indistinguishable from BBR—see the left three columns in Table 3.3. We next applied the RMMP algorithm to this dataset. Examining the size of the first active set reveals setting $k \approx 3000$, would give exactly the same results as the MP algorithm—see typical effects of changing k in Table 3.4 for $\gamma = 10$. We point out that this is a huge reduction in the worst case memory required, an approximately 98% reduction

Probit link function, $\gamma = 10$				Logistic link function, $\gamma = 100$			
EM	**MP**			**BBR**	**MP**		
	$z = 1$	$z = 2$	$z = 3$		$z = 1$	$z = 2$	$z = 3$
0.252	0.207	0.250	0.252	0.178	0.168	0.178	0.178
0.764	0.614	0.755	0.764	0.450	0.422	0.450	0.450
-0.318	-0.263	-0.314	-0.318	-0.124	-0.1161	-0.124	-0.124
0.834	0.667	0.824	0.834	0.713	0.666	0.712	0.713
0.894	0.719	0.884	0.894	0.656	0.613	0.655	0.656
-0.304	-0.243	-0.301	-0.304	0	0	0	0
-0.782	-0.627	-0.773	-0.782	-0.511	-0.477	-0.510	-0.511
-0.039	-0.037	-0.039	-0.039	0	0	0	0
-0.036	-0.029	-0.036	-0.036	0	0	0	0
0	0	0	0	0	0	0	0
-0.327	-0.266	-0.324	-0.327	-0.030	-0.028	-0.030	-0.030
L_1 Norm	0.878	0.050	0		0.172	0.003	0

Table 3.2: Table with columns showing values of the MAP estimates of $\boldsymbol{\beta}$ obtained by the batch algorithms (EM on the left half, for probit regression and BBR on the right half for logistic regression) and three successive iterates of the MP algorithm applied to the simulated dataset. The final row displays the L_1 norm of the difference between the batch algorithm estimates (EM or BBR as appropriate) and the MP algorithm estimates. The results shown here are representative of those obtained for other values of γ as well.

($k = 3000$ vs. $d = 21989$ originally). Note also that the size of k should be compared relative to the nonzero components for MAP $\boldsymbol{\beta}$ (150 and 31 for $\gamma = 10$ and $\gamma = 100$ respectively).

We further test the limits of the algorithm, by running it with $k = 300$ for $\gamma = 100$. The RMMP algorithm performs very well, requiring about $z = 7$ passes (only two more than the MP algorithm) to converge to correct parameter values. For $\gamma = 10$, where $k = 300$ is small (only twice the number of non-zero components in the MAP $\boldsymbol{\beta}$), once again the same kind of results hold, with the MP algorithm needing about 7 passes over the dataset and the RMMP algorithm needing about 15 passes to converge to the batch $\boldsymbol{\beta}$.

Finally, we present results of application of the algorithms to the RCV1-v2 data sets. For the RCV1-v2 training data ($d = 47,152$, $t = 23,149$), sparsity enables application of BBR to obtain the batch MAP $\boldsymbol{\beta}$ parameter values, as well as the MP algorithm, although this is quite cumbersome. For $\gamma = 10$ (a fairly high amount of regularization), we find essentially the same qualitative results as the ModApte dataset—it takes about $z = 6$ passes through the dataset to obtain indistinguishable parameter values as BBR.

j	**BBR**	**MP**		**RMMP**, $k = 300$		
		$z = 3$	$z = 5$	$z = 3$	$z = 5$	$z = 7$
1	-1.588	-1.527	-1.586	-1.451	-1.573	-1.588
9	1.188	0.957	1.185	0.688	1.143	1.188
13	0.847	0.793	0.846	0.678	0.839	0.847
147	0.813	0.795	0.813	0.696	0.812	0.813
31	0.801	0.618	0.800	0.356	0.772	0.801
⋮	⋮	⋮	⋮	⋮	⋮	⋮
3	-2.264e-2	-2.127e-2	-2.240e-2	-3.247e-2	-2.062e-2	-2.264e-2
62	-1.757e-2	-2.840e-2	-1.779e-2	-3.346e-2	-2.045e-2	-1.757e-2
2	1.552e-2	1.542e-2	1.548e-2	1.610e-2	1.525e-2	1.552e-2
12	-1.467e-2	-1.956e-2	-1.480e-2	-1.415e-2	-1.643e-2	-1.467e-2
8	1.277e-2	2.870e-2	1.320e-2	2.915e-2	1.776e-2	1.278e-2
L_1 Norm		1.691	0.034	3.496	0.4027	3e-4

Table 3.3: Results obtained on the ModApte dataset. The 5 highest and 5 lowest magnitude non-zero coefficients of MAP $\boldsymbol{\beta}$ for $\gamma = 100$ are shown. In table are the indices of $\boldsymbol{\beta}$, coefficients from BBR and those obtained after a particular number of passes over the data using the MP algorithm (full memory) and the RMMP algorithm with $k = 300$.

j	**BBR**	**RMMP**, $z = 8$				
		$k = 3120^*$	$k = 2000$	$k = 1000$	$k = 600$	$k = 300$
292	2.695	2.695	2.695	2.695	2.695	2.699
20	2.268	2.268	2.260	2.260	2.259	2.273
1	-2.010	-2.010	-2.009	-2.009	-2.009	-2.005
341	1.755	1.755	1.754	1.754	1.754	1.742
147	1.572	1.572	1.572	1.572	1.572	1.568
⋮	⋮	⋮	⋮	⋮	⋮	⋮
66	4.943e-3	4.944e-3	4.849e-3	4.821e-3	4.849e-3	3.224e-3
134	4.488e-3	4.479e-3	4.720e-3	4.756e-3	4.712e-3	1.677e-2
267	-2.057e-3	-2.068e-3	-1.863e-3	-1.897e-3	-1.852e-3	-3.427e-3
28	-1.652e-3	-1.644e-3	-1.542e-3	-1.560e-3	-1.537e-3	-1.991e-3
56	-1.518e-4	-1.540e-4	-3.059e-4	-2.983e-4	-3.121e-4	-8.640e-4
L_1 Norm		1.4e-3	0.047	0.044	0.048	1.269

Table 3.4: Results for the ModApte dataset: Illustrating the effect of changing k. The 5 highest and 5 lowest magnitude non-zero coefficients of MAP $\boldsymbol{\beta}$ for $\gamma = 10$ are shown. In table are the indices of $\boldsymbol{\beta}$, coefficients from BBR and those obtained after 8 passes over the data using the RMMP algorithm. * For $k = 3120$, RMMP behaves the same as the MP algorithm.

$\gamma = 10$					$\gamma = 100$, $k = 2500$	
$\boldsymbol{\beta}$ index	**BBR**	**RMMP,** $k = 1500$			$\boldsymbol{\beta}$ index	**RMMP**
		$z = 2$	$z = 5$	$z = 10$		$z = 10$
12220	18.065	0	18.084	18.065	12220	17.234
27407	9.909	7.988	9.904	9.909	37665	-11.901
37665	-8.201	-2.255	-8.118	-8.201	43626	8.654
46160	7.144	4.061	7.133	7.144	27407	8.308
5946	6.453	5.142	6.436	6.453	5946	8.215
33192	-6.211	-2.066	-6.159	-6.211	19647	6.326
43626	6.164	4.430	6.157	6.164	39539	5.782
21160	5.661	3.498	5.644	5.661	29641	4.728
19647	5.573	6.587	5.539	5.573	37471	4.621
29641	5.472	4.810	5.473	5.472	41148	4.507
L_1 Norm		87.940	0.480	0.001		

Table 3.5: RCV1-v2 results. Left portion RCV1-v2 training dataset, right BIG-RCV dataset.

The RMMP algorithm also gives excellent results in about 10 passes, see the left portion of Table 3.5 with $k = 1500$.

For the BIG-RCV dataset ($d = 288,062$, $t = 421,816$) however, computational and memory limitations made it impossible to run the batch algorithms on this dataset (also the MP algorithm). It is precisely for cases like this that the RMMP algorithm is useful, and we were able to obtain parameter estimates for reasonable settings of regularization in a few days—see for example, the right portion of Table 3.5.

In order to further make the case for examining massive datasets in their entirety, we performed the following experiment: We obtained the best possible predictive parameters using 10-fold cross-validation on the RCV1-v2 training dataset. This is an expensive computation, involving many repeated BBR training instances for many different values of the regularization parameter (we searched over $\gamma = 0.01, 0.1, 1, 10, 100$) and takes about a day. The final cross-validation chosen $\boldsymbol{\beta}$ has 1010 non-zero parameters.

We then trained a separate sparse logistic classifier on the BIG-RCV dataset using the RMMP algorithm with $k = 3000$ and $\gamma = 40$. Setting $\gamma = 40$ results in 1015 non-zero MAP $\boldsymbol{\beta}$ coefficients which is approximately the same number of non-zero coefficients as the cross-validation chosen $\boldsymbol{\beta}$. This is also an expensive computation requiring a

	"Optimized" $\boldsymbol{\beta}$ trained on RCV1-v2 training data		"Naive" $\boldsymbol{\beta}$ trained on BIG-RCV	
	Relevant	Not Relevant	Relevant	Not Relevant
Retrieved	38,821	7,415 (**83.96**%)	40,655	6,017 (**87.11**%)
Not Retrieved	16,368 (**70.34**%)	319,994	14,534 (**73.67**%)	321,392

Table 3.6: This table shows confusion matrices for prediction results on the RCV Test dataset. The CV $\boldsymbol{\beta}$ (trained on the RCV1-v2 training data set) results are on the left and the MAP $\boldsymbol{\beta}$ (trained on the BIG-RCV dataset, with $\gamma = 30$, $k = 3000$) results are on the right. Also shown are recall and precision percentages in bold and brackets. There are approximately 383,000 examples in the test dataset.

few days. Finally, we compare the predictive accuracy of both classifiers on the unused RCV test set (comprising the unused two portions of the original RCV1-v2 test data).

The results, shown in Table 3.6, demonstrate that using the information in extra examples, the "unsophisticated" classifier trained on the much larger dataset outperforms the "optimized" classifier trained on a smaller dataset.

3.8 Conclusions

In this chapter we present an asymptotically convergent online algorithm that builds sparse generalized linear models for massive datasets. We also present efficient multi-pass algorithms that examine observations sequentially and thus enable learning on massive datasets. Both algorithms exploit sparsity of input data. We effectively applied the algorithms to very large, sparse datasets, for which state-of-the-art batch algorithms are impractical/cumbersome, and our results show that examining such datasets in their entirety can lead to better classifier performance.

Some areas of further research that this work opens up are: extension of the algorithms for a hierarchical prior model so that the choice of regularization is less important, the possible application of our methods to kernel classifiers, and applications to multi-class classification problems.

.8

3.A Log-likelihood Taylor Approximations

Here we show the Taylor expansions for the quadratic approximations to the log-likelihood function. To simplify notation, let $c(\boldsymbol{\beta}) = \boldsymbol{\beta}^T \mathbf{x}_i$ and $\hat{c} = \boldsymbol{\beta}_{i-1}^T \mathbf{x}_i$. The link function (we will restrict analytical results to the logistic and probit link functions) is $\Phi(z)$ as before and we denote its first and second derivative, with respect to z, by $\Phi'(z)$ and $\Phi''(z)$ respectively.

Consider the case where $y_i = 1$:

$$\begin{aligned}
\log \Phi(c) &\approx \log \Phi(\hat{c}) + (c - \hat{c}) \frac{\Phi'(\hat{c})}{\Phi(\hat{c})} + \frac{(c-\hat{c})^2}{2} \left(\frac{\Phi''(\hat{c})}{\Phi(\hat{c})} - \left(\frac{\Phi'(\hat{c})}{\Phi(\hat{c})} \right)^2 \right) \\
&\propto \frac{\Phi'(\hat{c})}{\Phi(\hat{c})} c + \frac{1}{2} \left(\frac{\Phi''(\hat{c})}{\Phi(\hat{c})} - \left(\frac{\Phi'(\hat{c})}{\Phi(\hat{c})} \right)^2 \right) c^2 - \hat{c} \left(\frac{\Phi''(\hat{c})}{\Phi(\hat{c})} - \left(\frac{\Phi'(\hat{c})}{\Phi(\hat{c})} \right)^2 \right) c
\end{aligned}$$

so that:

$$a_i = \frac{1}{2} \left(\frac{\Phi''(\hat{c})}{\Phi(\hat{c})} - \left(\frac{\Phi'(\hat{c})}{\Phi(\hat{c})} \right)^2 \right)$$

and

$$b_i = \frac{\Phi'(\hat{c})}{\Phi(\hat{c})} - \hat{c} \left(\frac{\Phi''(\hat{c})}{\Phi(\hat{c})} - \left(\frac{\Phi'(\hat{c})}{\Phi(\hat{c})} \right)^2 \right).$$

Analogously, when $y_i = 0$:

$$\log(1 - \Phi(c)) \approx \log(1 - \Phi(\hat{c})) - (c - \hat{c}) \frac{\Phi'(\hat{c})}{1 - \Phi(\hat{c})} - \frac{(c-\hat{c})^2}{2} \left(\frac{\Phi''(\hat{c})}{1 - \Phi(\hat{c})} + \left(\frac{\Phi'(\hat{c})}{1 - \Phi(\hat{c})} \right)^2 \right)$$

so that:

$$a_i = -\frac{1}{2} \left(\frac{\Phi''(\hat{c})}{1 - \Phi(\hat{c})} + \left(\frac{\Phi'(\hat{c})}{1 - \Phi(\hat{c})} \right)^2 \right)$$

and

$$b_i = -\frac{\Phi'(\hat{c})}{1 - \Phi(\hat{c})} + \hat{c} \left(\frac{\Phi''(\hat{c})}{1 - \Phi(\hat{c})} + \left(\frac{\Phi'(\hat{c})}{1 - \Phi(\hat{c})} \right)^2 \right).$$

For the probit link function:

$$\begin{aligned}\Phi(z) &= \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx \\ \Phi'(z) &= \frac{1}{\sqrt{2\pi}} e^{-z^2/2} \\ \Phi''(z) &= \frac{-z}{\sqrt{2\pi}} e^{-z^2/2},\end{aligned}$$

whereas for the logistic link function:

$$\begin{aligned}\Phi(z) &= \frac{e^z}{1+e^z} \\ \Phi'(z) &= \frac{e^z}{(1+e^z)^2} \\ \Phi''(z) &= \frac{(e^z)(1-e^z)}{(1+e^z)^3}.\end{aligned}$$

These expressions then allow us to compute the a_i, b_i in the cases needed.

3.B Derivation of the Shooting Algorithm

In this appendix we derive the modified Shooting algorithm, Algorithm 1 and discuss its efficient implementation. We derive Shooting by performing a simple subgradient analysis of the system. Reviewing concepts very briefly, the subgradient of a convex function $f(\beta)$ is defined as:

$$\partial f(\beta) = \{s | f(\beta + \Delta) \geq f(\beta) + s|\Delta, \forall \Delta \in \mathbb{R}\}.$$

i.e. the subgradient is the entire range of slopes s, such that a line through $(\beta, f(\beta))$ with slope s contains f in its upper half-space. This is a generalization of the derivative which reduces to the derivative, $\partial f(\beta) = \frac{\partial f(\beta)}{\partial \beta}$ whenever f is differentiable. As a simple example, the subgradient of $f(\beta) = |\beta|$, the absolute value function (which is

non-differentiable at $\beta = 0$) is:

$$\partial f(\beta) = \begin{cases} \{-1\}, & \beta < 0 \\ [-1, 1], & \beta = 0 \\ \{1\}, & \beta > 0 \end{cases}$$

As one expects, analogous to optimality conditions resulting from setting the gradient of a differentiable function to zero, optimality conditions for non-differentiable functions result from restrictions on the subgradient. In particular we appeal to the following result from non-smooth analysis:

Theorem *$\hat{\beta}$ is a global minimizer of a convex function $f(\beta)$ if and only if $0 \in \partial f(\hat{\beta})$.*

Now to our particular problem. We need to find $\boldsymbol{\beta}$ that is a solution to:

$$\max_{\boldsymbol{\beta}} \left(\boldsymbol{\beta}^T \boldsymbol{\Psi} \boldsymbol{\beta} + \boldsymbol{\beta}^T \boldsymbol{\theta} - \gamma \|\boldsymbol{\beta}\|_1 \right).$$

The convexity of the problem allows us to make incremental progress towards the maxima coordinate-wise. Computing the j'th component of the subgradient of the function (keeping all other components fixed) we get:

$$\begin{aligned} & \tfrac{\partial}{\partial \beta_j}(\boldsymbol{\beta}^T \boldsymbol{\Psi} \boldsymbol{\beta}) + \tfrac{\partial}{\partial \beta_j}(\boldsymbol{\beta}^T \boldsymbol{\theta}) - \gamma \partial(\textstyle\sum_{j=1}^{d}(|\beta_j|) \\ = \quad & 2(\boldsymbol{\Psi}\boldsymbol{\beta})_j + \theta_j - \gamma \partial(|\beta_j|) \\ = \quad & 2\Psi_{jj}\beta_j + 2(\boldsymbol{\Psi}'\boldsymbol{\beta})_j + \theta_j - \gamma \partial(|\beta_j|) \end{aligned}$$

where $(\boldsymbol{\Psi}'\boldsymbol{\beta})_j$ is the j'th component of the vector $\boldsymbol{\Psi}'\boldsymbol{\beta}$ and Ψ_{jj} refers to the (j, j)'th element of the matrix $\boldsymbol{\Psi}$. The second equation follows from the first as the subgradient of a differentiable function is just its derivative and since matrix $\boldsymbol{\Psi}$ is symmetric (it is just a weighted sum of outer products). Now if we plug in the subgradient of the non-differentiable absolute value function, setting $\Omega_j = 2(\boldsymbol{\Psi}'\boldsymbol{\beta})_j + \theta_j$, we obtain the

subgradient of the optimization problem (whose j'th component we denote by ∂_{β_j}) as:

$$\partial_{\beta_j} = \begin{cases} \{2\Psi_{jj}\beta_j + \Omega_j \ + \gamma\}, & \beta_j < 0 \\ [\Omega_j \ - \gamma, \Omega_j \ + \gamma], & \beta_j = 0 \\ \{2\Psi_{jj}\beta_j + \Omega_j \ - \gamma\}, & \beta_j > 0 \end{cases}$$

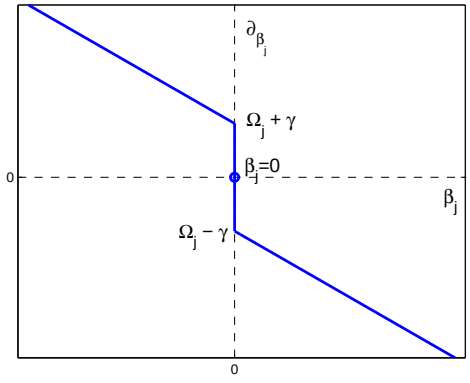
This is a piecewise linear function with fixed negative slope $2\Psi_{jj}$ and a constant jump of fixed size 2γ at $\beta_j = 0$ (Ψ_{jj} can be proven to always be negative by looking at the update formula for $\mathbf{\Psi}$ and using the fact that $\forall i, a_i < 0$). Using the optimality criteria (now for maximization since $-|\beta_j|$ is a concave function) naturally leads to the modified Shooting algorithm, illustrated in Figure 3.7.

Now to questions regarding the efficient implementation of this algorithm. In the modified Shooting algorithm, after each component update (change in β_j) we need to modify $\mathbf{\Omega}$ (the update $\mathbf{\Omega}$ step in the algorithm). This can be implemented efficiently using the following result (similar to the trick detailed in Minka, 2001):

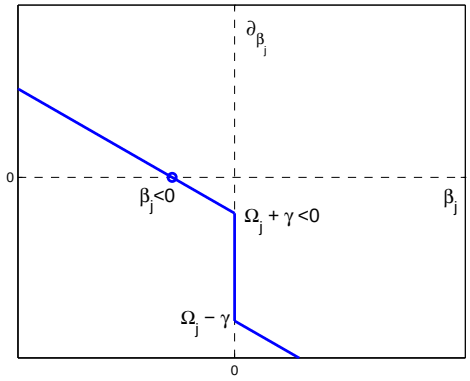
$$\mathbf{\Omega}^{new} = \mathbf{\Omega}^{old} + 2\mathbf{\Psi}'_{(.j)}(\Delta\beta_j)$$

where $\Delta\beta_j$ is the change in β_j and $\mathbf{\Psi}'_{(.j)}$ is the j'th column of $\mathbf{\Psi}'$. Thus each component update of Shooting can be done in $O(d)$ computational time. Now, if as before the maximum number of non-zero components of $\boldsymbol{\beta}$ along the solution path to MAP $\boldsymbol{\beta}$ is m, only m such updates will need to be made, giving a total time requirement per iteration of $O(md)$.

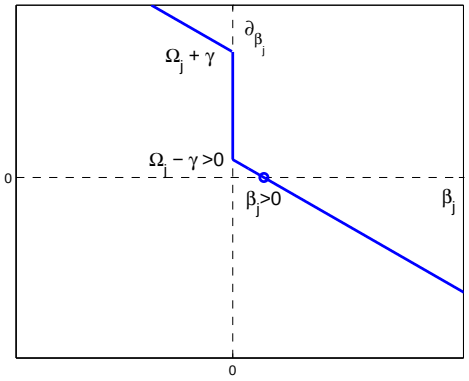
Finally we detail how to carry out the $\mathbf{\Omega}$ update in small space given we don't explicitly store the complete matrix $\mathbf{\Psi}$, but rather only a submatrix of it $\tilde{\mathbf{\Psi}}$, based on the active set of components S, as required in Algorithm 3. Note that since we are discussing the MP algorithm, the location where we take the quadratic approximation, $\boldsymbol{\beta}_{i-1}$, is constant throughout the pass through the fixed dataset, D_t. We exploit the fact that for constant $\boldsymbol{\beta}_{i-1}$, you don't explicitly need the matrix $\mathbf{\Psi}$ (or $\tilde{\mathbf{\Psi}}$) to determine



(a)



(b)



(c)

Figure 3.7: Illustration of cases occurring in the Shooting algorithm (a) If $|\Omega_j| \leq \gamma$ the constant portion of the subgradient contains zero. In this case, set $\beta_j = 0$ (b) If instead, $\Omega_j < -\gamma$, the optimality conditions will be satisfied by setting $\beta_j = \frac{-\gamma - \Omega_j}{2\Psi_{jj}}$ (c) The case analogous to (b) but when $\Omega_j > \gamma$. Here the subgradient is set equal to zero when $\beta_j = \frac{\gamma - \Omega_j}{2\Psi_{jj}}$.

$\mathbf{\Omega}$. Indeed, after going through all the observations in the dataset (pass z, say):

$$\mathbf{\Omega} = 2\mathbf{\Psi}'\boldsymbol{\beta}_{z-1} + \boldsymbol{\theta} = 2\left(\sum_{i=1}^{t} a_i \left(\mathbf{x}_i\mathbf{x}_i^T - \text{diag}(\mathbf{x}_i^2)\right)\right)\boldsymbol{\beta}_{z-1} + \sum_{i=1}^{t} b_i\mathbf{x}_i,$$

which follows from the definition of $\mathbf{\Omega}, \boldsymbol{\theta}$ and $\mathbf{\Psi}'$. In the above equation, $\text{diag}(\mathbf{x}_i^2)$ is a $d \times d$ matrix zero everywhere except the diagonal entries, which consists of the elements of the vector $\mathbf{x}_i$ squared component-wise. This leads to the following equation for $\mathbf{\Omega}$:

$$\mathbf{\Omega} = 2\sum_{i=1}^{t} a_i(\boldsymbol{\beta}_{z-1}^T\mathbf{x}_i)\mathbf{x}_i - 2\sum_{i=1}^{t} a_i(\mathbf{x}_i^2\boldsymbol{\beta}_{z-1}) + \sum_{i=1}^{t} b_i\mathbf{x}_i,$$

where $(\mathbf{x}_i^2\boldsymbol{\beta}_{z-1})$ is a vector whose entries are $\mathbf{x}_i^2$ multiplied by $\boldsymbol{\beta}_{z-1}$ component-wise. Note the first sum is just a weighted combination of the input data ($\boldsymbol{\beta}_{z-1}^T\mathbf{x}_i$ is a scalar). Thus, our final update formula results:

$$\mathbf{\Omega}^{new} = \mathbf{\Omega}^{old} + (2a_i\boldsymbol{\beta}_{z-1}^T\mathbf{x}_i + b_i)\mathbf{x}_i - 2a_i(\mathbf{x}_i^2\boldsymbol{\beta}_{z-1}).$$

As can be seen, computing this update per observation takes time and space $O(d)$, and having restricted the number of non-zero components of $\boldsymbol{\beta}$ to k, a total computational cost per iteration of Shooting to $O(kd)$.

3.C Non-divergence Proof Sketch

We present a proof sketch for the non-divergence of the online algorithm in the infinite data limit. The intuition for convergence to the optima is as follows: as $t \to \infty$, the Bayesian central limit theorems dictate that the posterior distribution tends (in distribution) to a multivariate Normal (with ever shrinking covariance, Bernardo and Smith 1994). Thus, fewer and fewer parameters are required to encode the posterior distribution as more and more data is added (in the limit, this only the vector of $\boldsymbol{\beta}_{MLE}$, the maximum likelihood value of the parameters).

Suppose now that the online algorithm converges to a particular fixed point. In the infinite data limit, an infinite number of term approximations are taken at this fixed

point. Now, our Taylor polynomial based approximation preserves both the function value and its derivative, and an infinite number of approximations are jointly maximum at this fixed point. This implies the fixed point is an optima of the posterior distribution.

Thus, if the approximation converges to a fixed point, it is the correct optima location. The above is a modification of the fixed point Lemma in the paper on Laplace Propagation (Smola et al., 2003). One can also prove unbiasedness which follows from our update rules and a minor modification of a theorem in Opper, (1999). Even though Opper derives his results based on a Gaussian prior on the parameters $\boldsymbol{\beta}$ (corresponding to L_2 regularization), the general format of Opper's theorem is still applicable in our case because, in the infinite data limit, the prior is inconsequential.

References

J. M. Bernardo and A. F. M. Smith. *Bayesian Theory*. John Wiley and Sons, Inc., 1994.

S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM Journal on Scientific Computing*, 20(1):33–61, January 1999.

B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *The Annals of Statistics*, 32(2):407 – 499, 2004.

M. A. T. Figueiredo and A. K. Jain. Bayesian learning of sparse classifiers. 1:35 – 41, 2001.

W. J. Fu. Penalized regressions: The bridge versus the lasso. *Journal of Computational and Graphical Statistics*, 7(3):397–416, 1998.

A. Genkin, D. D. Lewis, D. Madigan, S. Eyheramendy, and W.-H. Ju. Sparse bayesian classifiers for text categorization, April 30 2003. URL `http://www.stat.rutgers.edu/ madigan/PAPERS/shortFat-v13.pdf`.

T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu. The entire regularization path for the support vector machine. *Journal of Machine Learning Research*, 5:1391 – 1415, 2004.

T. Jaakkola and M. Jordan. Bayesian parameter estimation via variational methods. *Statistics and Computing*, 10:25 – 37, 2000.

R. E. Kass and A. E. Raftery. Bayes factors. *Journal of the American Statistical Association*, 90:773 – 795, 1995.

B. Krishnapuram, L. Carin, M. A. T. Figueiredo, and A. J. Hartemink. Sparse multinomial logistic regression: Fast algorithms and generalization bounds. *IEEE Transactions on Pattern Analalysis and Machine Intelligence*, 2005. To appear.

D. D. Lewis. Reuters-21578 text categorization test collection: Distribution 1.0 readme file (v 1.3), 2004. URL `http://www.daviddlewis.com/resources/testcollections/reuters21578/readme.txt`.

D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361 – 397, 2004.

D. J. C. MacKay. Probable networks and plausible predictions: a review of practical bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6:469 – 505, 1995.

T. P. Minka. Expectation propagation for approximate bayesian inference. In Jack Breese and Daphne Koller, editors, *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI-01)*, pages 362–369, San Francisco, CA, August 2–5 2001a. Morgan Kaufmann Publishers.

T. P. Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 2001b.

A. Moore and P. Komarek. Logistic regression for data mining and high-dimensional classification, April 29 2004. URL `http://citeseer.ist.psu.edu/652603.html`.

M. Opper. A bayesian approach to on-line learning. Technical report, 1996. NCRG/99/029.

M. R. Osborne, B. Presnell, and B. A. Turlach. A new approach to variable selection in least squares problems. *IMA Journal of Numerical Analysis*, 20(3):389–403, July 2000.

Y. Qi, T. P. Minka, R. W. Picard, and Z. Ghahramani. Predictive automatic relevance determination by expectation propagation. In *Proceedings of Twenty-first International Conference on Machine Learning*, Banff, Alberta, Canada, July 4-8 2004.

S. K. Shevade and S. S. Keerthi. A simple and efficient algorithm for gene selection using sparse logistic regression. *Bioinformatics.*, 19(17):2246 – 2253, 2003.

S. A. Solla and O. Winther. Optimal perceptron learning: an online bayesian approach. In D. Saad, editor, *Proceedings of the Newton Institute Workshop on On-Line Learning*, pages 379 – 398. Cambridge University Press, 1998.

R. J. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58(1):267–288, 1996.

T. Zhang. On the dual formulation of regularized linear systems. *Machine Learning*, (46):91–129, 2002.

T. Zhang and F. J. Oles. Text categorization based on regularized linear classification methods. *Information Retrieval*, 4(1):5–31, 2001.

Chapter 4

Finding Predictive Runs for Structured Data Classification[1]

4.1 Predictive Runs

In this chapter and the next, we consider regression and classification problems where the predictor variables are ordered and naturally form groups (structured data classification problems). For example, in predicting whether or not a vaccinated animal survives an anthrax challenge, relevant attributes might include a toxin neutralization assay (TNA) measured at ten different time points (i.e., a group of ten predictor variables), a protective antigen assay, measured at 20 time points (i.e., a group of twenty predictor variables), and vaccine dilution (i.e., a group containing a single predictor variable). We believe that many regression and classification applications exhibit such structure and we describe several in what follows.

Standard modelling approaches that ignore the group structure or ordering can lead to models that provide good predictive performance but make little sense. For example, applying feature selection to the problem above might result in selecting TNA predictor variables corresponding to measurements at weeks 2, 8, and 48 and dropping the measurements at weeks, 4, 6, 12, 20, 32, 40, and 52. Similarly, feature selection might result in selecting values of other assays at seemingly arbitrary timepoints. Since the assay measurements are generally serially correlated, small perturbations to the training data often lead to the selection of a drastically different set of predictor variables. We argue that in many applications, it makes more sense to select (or omit) contiguous "runs" of predictor variables - for example, TNA measurements from week 2 through week 8.

[1]The basic content in this chapter has appeared in "Finding Predictive Runs with LAPS", Balakrishnan S. and Madigan D., submitted manuscript, 2007.

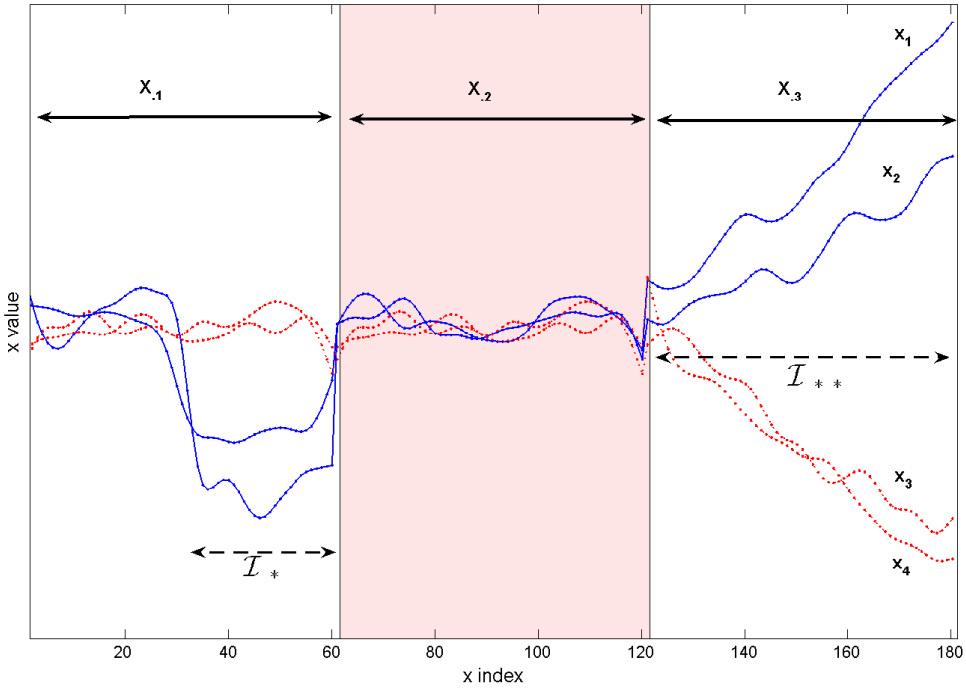


Figure 4.1: Typical classification problem setup. Plotted are 4 examples, two each drawn from the two different classes (shown in red dotted and blue solid lines). Also shown in the figure are the 3 different groups (the differently shaded and delineated bands $\mathbf{x}_{.1}, \mathbf{x}_{.2}, \mathbf{x}_{.3}$) and potential locations for predictive runs.

Again we restrict our focus to binary classification problems. For each example, given an input vector $\mathbf{x}_i = [\mathbf{x}_{i1}, \ldots, \mathbf{x}_{id}]$, we seek to predict the corresponding label $y_i \in \{-1, 1\}$. Each $\mathbf{x}_{ij}$ corresponds to a "group," $\mathbf{x}_{ig} = [x_{ig}^{(1)}, x_{ig}^{(2)}, \ldots, x_{ig}^{(T_g)}], T_g \geq 1$(the group length). Figure 1 shows a problem with three equal-size groups, $\mathbf{x}_{.1}, \mathbf{x}_{.2}$ and $\mathbf{x}_{.3}$, where the second half of the first group (variable indices $\sim$ 30—60, denoted by $\mathcal{I}_*$) has some discriminatory power with respect to the two classes (indicated by blue-solid and red-dotted lines), the second group has no discriminatory power, and the entire third group (indices $\mathcal{I}_{**}$), is useful, with the possible exception of the first few values.

We also continue to restrict our attention to linear logistic regression models. Within this hypothesis class, we seek to develop a modelling approach that automatically identifies the sub-group indices, or runs, $\mathcal{I}_*$ and $\mathcal{I}_{**}$ and the corresponding regression parameters.

4.2 Modelling Predictive Runs

Our work builds on two recent extensions to the original "Lasso" L_1-regularized regression models of Tibshirani (1996) [2]. The "fused Lasso" (Tibshirani et al., 2005) addresses problems like ours where the variables are ordered (their development and experiments are described for inputs with one group, i.e., $\mathbf{x} = \mathbf{x}_{\cdot 1}$). The fused Lasso encourages contiguous subgroups of **identical** coefficients (the corresponding variables being highly correlated) to be non-zero together. They accomplish this through an additional L_1 penalty (besides the regular Lasso regularization term) on the differences of successive coefficient values ($\beta_k - \beta_{k+1}$ terms). In a sense, what we propose below is a "soft" version of the fused lasso.

Our work is more closely related to another Lasso extension, the "group Lasso" (Yuan and Lin, 2006; Meier et al., 2006). Here the emphasis is on adapting the Lasso sparsity to **sets of predictors**. In particular, the group lasso either selects or omits entire groups of variables, where the data analyst pre-specifies what attributes form the groups. An elegant result of this formulation is that it reduces to the Lasso when all the variable sets are of size one.

Our proposed approach identifies runs (or contiguous subgroups) of model coefficients, that are similar (like a soft fused Lasso) and that will be selected together (have non-zero model coefficients en-block, like the group Lasso). The challenge is that we neither know the within-group run structure of the attributes, nor the amount of similarity within runs beforehand. The following sections outline our approach to these problems, which essentially consists of modifying the group Lasso penalty to potentially include similarity between coefficients and searching over group partitions into runs. We call this approach LAPS (the Lasso with Attribute Partition Search).

---

[2] The Lasso uses L_1-regularization to achieve simultaneous sparsity and complexity control. Genkin et al. (2004) and others report excellent predictive performance in high dimensional applications within this modelling framework.

4.2.1 The LAPS model

Given hyper-parameters λ (a regularization parameter) and k (a parameter governing serial correlation of run coefficients), LAPS finds logistic regression coefficients $\boldsymbol{\beta}$ and the run structure $\mathcal{I}$ such that:

$$\underset{\boldsymbol{\beta},\mathcal{I}}{\operatorname{argmin}}\, nll(\boldsymbol{\beta}) + \lambda \sum_{j=1}^{J} s_j \left\| \boldsymbol{\beta}_{\mathcal{I}_j} \right\|_{K_j}, \tag{4.1}$$

where $nll(\boldsymbol{\beta}) = -\sum_{i=1}^{t} \log \Phi(-y_i \boldsymbol{\beta}^T \mathbf{x}_i)$, is the negative log-likelihood involving the logistic link function $\Phi(z) = \frac{e^z}{1+e^z}$. The training data $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{t}$ comprises t labeled examples where the input examples, as before, are $\mathbf{x}_i = [\mathbf{x}_{i1}, \ldots, \mathbf{x}_{id}]$ and can be thought of compactly as a single p-dimensional vector (as presented in the introduction. Thus p = the total number of attributes = sum of the lengths of the d groups $= \sum_{g=1}^{d} T_g$). The run structure (or partition structure) $\mathcal{I}$ comprises the set of run indices $\mathcal{I}_j$, $\mathcal{I} = \{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_J\}$. These run indices, in turn, form a disjoint partition that covers the entire attribute index range (all groups), thus $\cup_{j=1}^{J} \mathcal{I}_j = 1 \ldots p$, and $\mathcal{I}_u \cap \mathcal{I}_v = 0, \forall u \neq v$. Additionally we impose the requirement that run indices respect all group boundaries (that is, runs never cross the boundaries between the various $\mathbf{x}_{\cdot j}$ for different js). The K_j matrices are positive definite matrices parameterized by a single scalar k (subsection 4.2.2 has details). The regularization term involves the K_j matrix norms[3] of the run coefficients (for a vector $\mathbf{z}$ and a matrix A, $\|\mathbf{z}\|_A = (\mathbf{z}^T A \mathbf{z})^{0.5}$). Finally, the $s_j = \sqrt{0.5(|\mathcal{I}_j| + 1)}$, are scalars factors that "normalize" the prior $\boldsymbol{\beta}$ variance (more about this also in subsection 4.2.2).

Figure 4.2 shows the resulting LAPS model applied to a small simulated example with two groups, where one entire group is discriminative. Whereas the Lasso results are unsatisfactory, as it finds only two non-zero coefficients among the discriminative group of correlated variables[4], LAPS does exactly what we'd like, finding runs in both

[3]This matrix norm penalty is the Mahalanobis distance of the run coefficients to the (appropriate-size) zero vector. In a Bayesian interpretation, zero is the location of the prior mode and the prior covariance matrix involves, K_j^{-1}. Details can be found in Appendix 4.A. Such matrix norms were suggested by Yuan and Lin (2006) (and in their references), but all their subsequent modelling and experiments used only the identity matrix.

[4]Since the Lasso models all coefficients as exchangeable and strongly favors parsimony, this behavior

regions, and giving (lower, but) almost equal predictive weight to the whole group (which is found to be a single run).

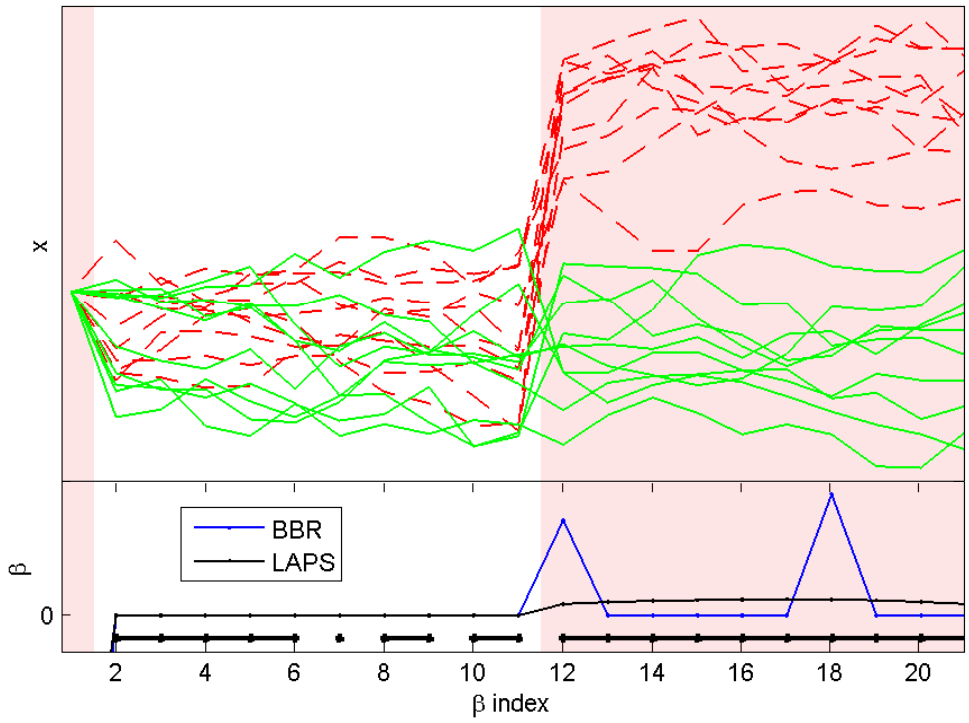


Figure 4.2: Simple example showing proof of concept. In the top portion we plot the data set used (created by collating different 10-dimensional highly correlated Gaussians as shown. The correlation boundaries define groups, which are shaded. 20 samples total, 10 from each class). The two classes are shown in different colors. The bottom portion of the plot shows the Lasso model coefficients (found using BBR Genkin et al. 2004 with hyperparameter selected by 10-fold CV) and the estimated LAPS model coefficients and inferred run structure, $\mathcal{I}$ (denoted by the bands below the coefficients). The thin shaded region on the left is for the intercept term, and its coefficient is cropped out of the y-axis of the bottom portion.

4.2.2 K—"Soft" Fusion

LAPS models use non-identity K matrices which provide them a very flexible set of modelling choices—all the way from strongly dependent run coefficients (via strong correlation structure of K) to models where the entire set of coefficients in the run is exchangeable ($K = \boldsymbol{I}$).

---

is expected.

We parameterize these K_j matrices by a single scalar k, based on the following assumptions. First, we assume that *a-priori* all the components of $\boldsymbol{\beta}$ have equal variance, regardless of the size of the run they will be in (a Bayesian interpretation is involved, see Appendix 4.A). Second, since we seek runs on ordered variables, we try to impose the requirement that consecutive model coefficients in the same run should be similar. We accomplish this via tri-diagonal K_j's. The corresponding K_j^{-1} is a symmetric positive definite matrix with ones on the diagonal (this ensures the equal variance of components) and terms in decreasing geometric progression (multiplicative factor k) proportional to the distance from the diagonal (a Green's matrix).

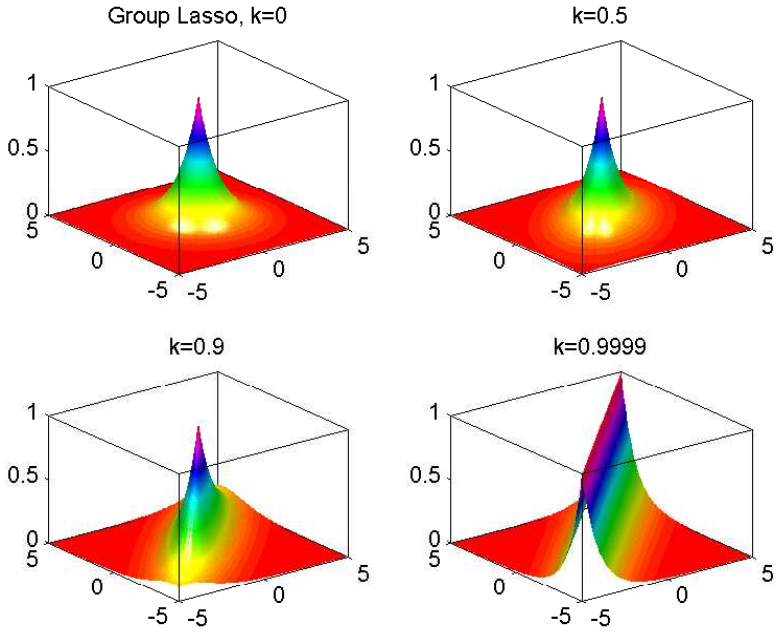


Figure 4.3: The effect of k on the prior for 2-D—illustrating soft fusion. Notice how as k increases, prior mass shifts favoring both parameters to be more like each other.

See Figure 4.3 for a graphical view of how the Bayesian prior varies with respect to the fusion parameter k on a two-variable size run. The k value rather intuitively controls how much soft fusion we enforce (for k=0, we obtain the group Lasso). For a slightly bigger example, consider the matrices obtained for a run of size 4, with k=0.5.

Here:

$$K^{-1} = \begin{pmatrix} 1 & 0.5 & 0.25 & 0.125 \\ 0.5 & 1 & 0.5 & 0.25 \\ 0.25 & 0.5 & 1 & 0.5 \\ 0.125 & 0.25 & 0.5 & 1 \end{pmatrix} \text{ and, } K = \begin{pmatrix} 1.333 & -0.667 & 0 & 0 \\ -0.667 & 1.667 & -0.667 & 0 \\ 0 & -0.667 & 1.667 & -0.667 \\ 0 & 0 & -0.667 & 1.333 \end{pmatrix}.$$

4.3 Learning LAPS models

We describe the algorithm for fitting full LAPS models (with parameters $\boldsymbol{\beta}$, $\mathcal{I}$, k, and λ) in stages. First, consider the situation where values for k, λ and the run structure $\mathcal{I}$ are known. In this case, given D, λ, $\mathcal{I}$ and k, we want to find $\boldsymbol{\beta}$ such that:

$$\underset{\boldsymbol{\beta}}{\operatorname{argmin}}\, g_{\lambda,\mathcal{I},k}(\boldsymbol{\beta}) = nll(\boldsymbol{\beta}) + \lambda \sum_{j=1}^{J} s_j \|\boldsymbol{\beta}_{\mathcal{I}_j}\|_{K_j}. \tag{4.2}$$

We will refer to this as the core LAPS optimization problem. It is a convex optimization problem and we use a standard block coordinate descent algorithm to solve it (see Algorithm 1, Meier et al. 2006. We use simple off-the-shelf line search and Newton solvers)[5]. Convexity is crucial, and the algorithm results from repeated application of the optimality criteria (Appendix 4.B provides a sketch of the derivation). Note that throughout we do not penalize the intercept term β_0. Next, consider the search for $\mathcal{I}$ (with λ and k still fixed).

4.3.1 $\mathcal{I}^*$—Run Structure Search

Motivated by the work of Consonni and Veronese (1995), we use a heuristic greedy procedure for the run structure search. The search starts at an initial $\mathcal{I}$ derived using the core LAPS optimization problem (see Appendix 4.D for details). The search then proceeds by locally perturbing an existing partition structure to generate a candidate run structure (see Figure 4.4). We then obtain the optimal $\boldsymbol{\beta}^*$ corresponding to this

[5]Although this is probably not the most efficient algorithm for this problem, in our experiments it proved to be quite reasonable.

Algorithm 4: Core LAPS optimization problem

Data: Training data D, initial $\boldsymbol{\beta}$.
Result: $\boldsymbol{\beta}$ that satisfies Equation 4.2
while *Convergence criteria not met* **do**
  $\beta_0 \leftarrow \text{argmin}_{\beta_0}\, g_{\lambda,\mathcal{I},k}(\boldsymbol{\beta})$ (Line search for intercept).
  for $j = 1, 2, \ldots, J$ **do**
    if $\left\|\nabla nll(\boldsymbol{\beta})_{\mathcal{I}_j}\right\|_{K_j^{-1}} \leq \lambda s_j$ **then**
      $\boldsymbol{\beta}_{\mathcal{I}_j} \leftarrow 0$
    else
      $\boldsymbol{\beta}_{\mathcal{I}_j} \leftarrow \text{argmin}_{\boldsymbol{\beta}_{\mathcal{I}_j}}\, g_{\lambda,\mathcal{I},k}(\boldsymbol{\beta}_{\mathcal{I}_j})$ (by Newton's method, say).
    end
  end
end

candidate. We use the optimal objective function value, $g_{\lambda,\mathcal{I},k}(\boldsymbol{\beta}^*)$ (Equation 4.2) to score the candidates. The search stops when all changes to the existing run structure result in worse scoring models. Note that our greedy search just involves repeated solutions of the (efficient) core LAPS optimization problem. Further, since the amount of regularization is fixed at this stage (λ, k constant), the optimal objective function value, g, makes a sensible scoring criterion for the models.

4.3.2 Selecting the Hyperparameters

Finally, we propose to search over a discrete grid for the remaining hyperparameters, λ and k. For each (λ,k) pair, we greedily search over the run structures for locally best $\mathcal{I}$ and $\boldsymbol{\beta}$ pairs (as outlined in subsection 4.3.1).

Having found $\boldsymbol{\beta}^*$ and $\mathcal{I}^*$ for every (λ,k) pair, we now score these locally "optimal" models. We cannot use the same g function value for scoring *across* the different grid points as they have different amounts of regularization. A cross-validation accuracy based score for instance, makes sense, but may prove computationally infeasible for even moderate size problems (this would require repeated solutions of the $\mathcal{I}^*$ search problem for each fold). We instead use an approximate marginal data likelihood based

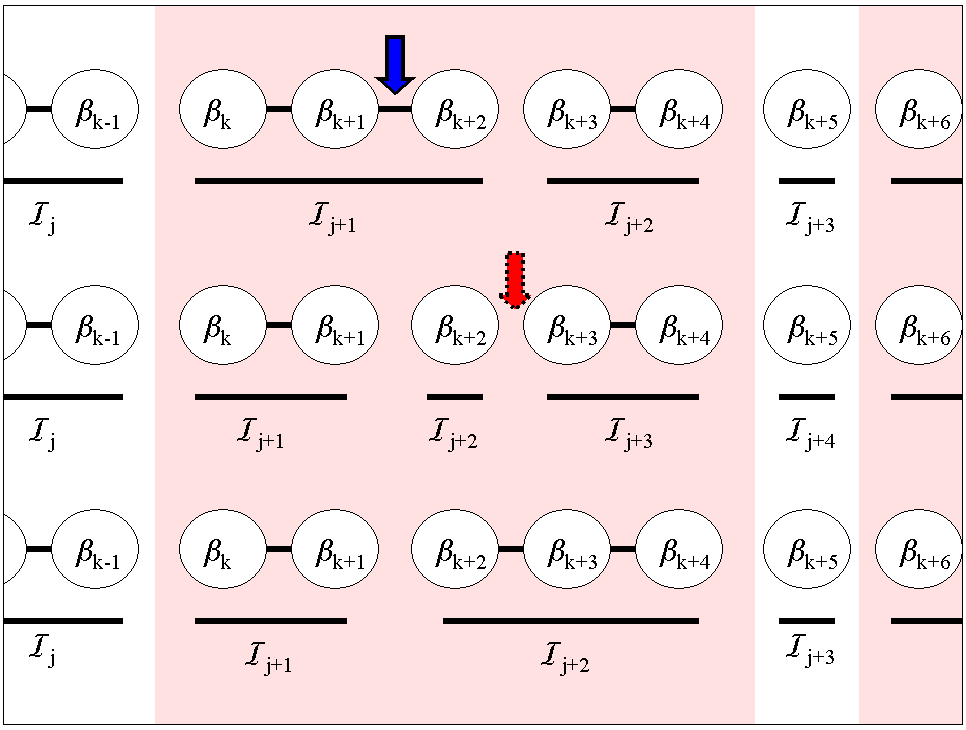


Figure 4.4: Illustrating the $\mathcal{I}^*$ search with a graphical representation of the model coefficients. Edges (parts of a run) can only occur between adjacent nodes (coefficients) within the same group (shaded regions, as before). The run structure $\mathcal{I}$, is the set of all connected components defining the runs, $\mathcal{I}_j$s. Our search strategy works by sequentially examining all potential or existing edges. Two accepted perturbations and the corresponding run partitions are shown for the middle group (from the top to bottom rows).

score (Appendix 4.C).

$$\begin{aligned} S(\boldsymbol{\beta}^*, \mathcal{I}^*, \lambda, k) &= nll(\boldsymbol{\beta}^*) + 0.5 \log |\Psi(\boldsymbol{\beta}^*, \mathcal{I}^*)| \\ &\quad + \lambda \sum_{j=1}^{J} s_j \|\boldsymbol{\beta}^*_{\mathcal{I}^*_j}\|_{K_j} - \sum_{j=1}^{J^*} \log \left(\frac{c(|\mathcal{I}^*_j|)}{|\Sigma_j|^{0.5}} \right) \\ &\quad - 0.5(nJ^* + 1) \log(2\pi), \end{aligned}$$

where $\Psi(\boldsymbol{\beta}, \mathcal{I}) = X^T A X + \lambda \sum_{j=1}^{J^*} s_j (\|\boldsymbol{\beta}_{\mathcal{I}_j}\|^{-1}_{K_j} K_j - \|\boldsymbol{\beta}_{\mathcal{I}_j}\|^{-3}_{K_j} B)$, with A being the $t \times t$ diagonal matrix formed by the $a_{ii} = \Phi(\boldsymbol{\beta}^{*T} \mathbf{x}_i)(1 - \Phi(\boldsymbol{\beta}^{*T} \mathbf{x}_i))$ terms, $B = \boldsymbol{b}\boldsymbol{b}^T$ (an outer product of $\boldsymbol{b} = K_j \boldsymbol{\beta}^*_{\mathcal{I}^*_j}$ vectors. The summation till J^* is only over the non-zero $\boldsymbol{\beta}^*$ run indices (and nJ^* is the number of non-zero $\boldsymbol{\beta}^*$ indices). Appendix 4.A has expressions

for c, Σ_j.

We compare the model scores $S(\boldsymbol{\beta}^*, \mathcal{I}^*, \lambda, k)$ over the entire (λ,k) grid and pick the best model amongst those (as defined, smaller S is better. This then results in values for λ^* and k^*). In our experiments we have found this procedure quite robust to variations in the dataset and parameter settings.

4.4 Experiments

We next apply LAPS models to real and simulated data. We are interested in evaluating both structural (run partitioning) performance and predictive accuracy. Predictive performance comparisons are made to Lasso logistic regression (using BBR[6], Genkin et al. 2004, publicly available software).

4.4.1 SIM Data

In our first set of experiments we simulate datasets from three different models, with regression coefficients designed to favor one of regular Lasso, the group Lasso, and LAPS. The data $\mathbf{x} \sim N(0,1)^{15}$, are simulated to be uncorrelated 15-dimensional Gaussian with mean zero and unit variance. The $\boldsymbol{\beta}_{true}$ are shown in Table 4.1. A large test dataset (10^4 examples) was also simulated for each set of regression coefficients. As can be seen, each set of coefficients favors one of the three models—the first set has no intentionally long runs (thus favors the Lasso model, SIM1), the second set has runs, but with no internal similarity (thus favors the group Lasso, SIM2), and the third has runs with extremely high similarity (SIM3). In all three cases, the coefficients are scaled such that the Bayes risk, $r = \sum_{i=1}^{t} \min\{\Phi(\boldsymbol{\beta}_{true}^T \mathbf{x}_i), 1 - \Phi(\boldsymbol{\beta}_{true}^T \mathbf{x}_i)\}$, on the large corresponding test dataset is 0.2. In order to assess sample size effects, we also simulate training datasets of two sizes, small (50 examples, denoted in the results by SM) and large (LG, 500 examples). There is only one group here, which corresponds to the entire set of predictors, [1—16] (Index 1 is for the intercept).

[6] `http://www.stat.rutgers.edu/~madigan/BBR`

Table 4.1: SIM data regression coefficients ($\boldsymbol{\beta}_{true}$, in columns). The first index corresponds is for the intercept term. The desired runs are shown as blocks in the columns (same as the blue bands at the bottom of the relevant plots in Figures 4.5,4.6 and 4.7).

$\boldsymbol{\beta}$ **Index**	**SIM1**	**SIM2**	**SIM3**
1	0	0	0
2	1.1500	0	0
3	0	-1.1609	-0.9540
4	0.5750	0.5804	-0.9540
5	-0.2875	-0.8706	-0.9540
6	0	0.5804	-0.9540
7	0	0	0
8	-0.2875	0	0
9	0.5750	0	0
10	0	-0.5804	-0.4770
11	1.1500	0.2902	-0.4770
12	0	-1.1609	-0.4770
13	-1.1500	0	0
14	0	0	0
15	0	0.8706	0.7155
16	-0.8625	-0.2902	0.7155

The results show that LAPS does a fairly reasonable job of finding non-zero coefficient runs[7] even with the small datasets, and performs much better on the large datasets (see the $\mathcal{I}$ bands in the plots in the figures). The inferred k^* parameter also provides insight, being 0 and 0.99 for SIM2-LG and SIM3-LG, indicating run structure that is not-at-all and highly similar respectively. Also, predictively, LAPS performs competitively with the Lasso (see Table 5.2[8]), having small gains and losses in the expected cases.

[7]We point out here that the zero coefficient runs aren't necessarily grouped because the the $\mathbf{x}$s are uncorrelated and g, is insensitive to zero-coefficients being grouped in a run together (MAP zero coefficients in any number of runs result in the same g value).

[8]The error estimates are obtained from the test set for the SIM data (SM/LG 1...3) and by 10-fold CV on BF and NHP(at k^*, V^* with standard error shown). $V = 2/\lambda^2$, is an equivalent paramterization of λ. V^* (and k^*, for LAPS) are found through grid search. The grid for k is consistently set to five values uniform over the range [0—0.99] (including both end points). The search ranges for V for Lasso and LAPS are [0.01—1] and [0.1—0.9] respectively for the SIM datasets . For the BF dataset, the ranges were [0.01—10^3] and [0.01—1], and [0.05—1], [0.1—0.9] for the NHP data (Lasso and then LAPS respectively). The Lasso V grid was always chosen at least thrice as fine as the uniform 10 grid points from the interval for LAPS.

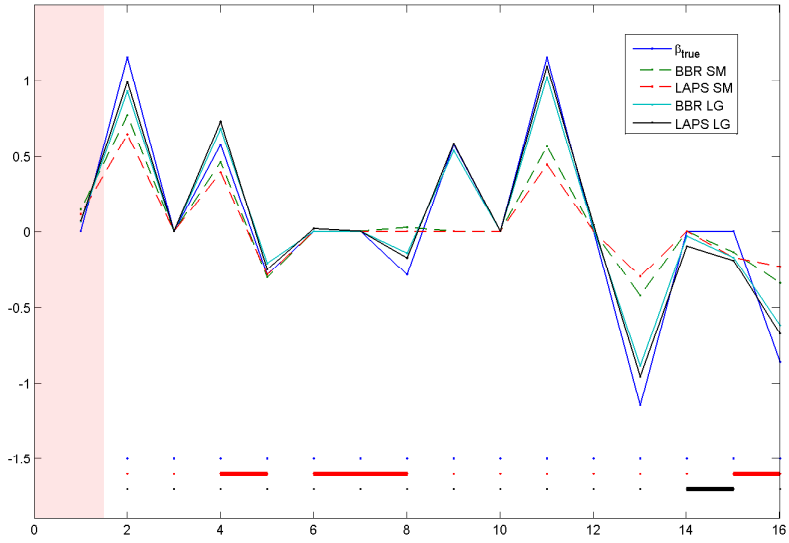


Figure 4.5: Lasso vs. LAPS on the SIM 1 dataset. The plots show the true, Lasso and best LAPS $\boldsymbol{\beta}$s and LAPS $\mathcal{I}$ (the bands at the bottom of each plot). The intended/true run structure for the SIM 1 data is also shown in blue bands.

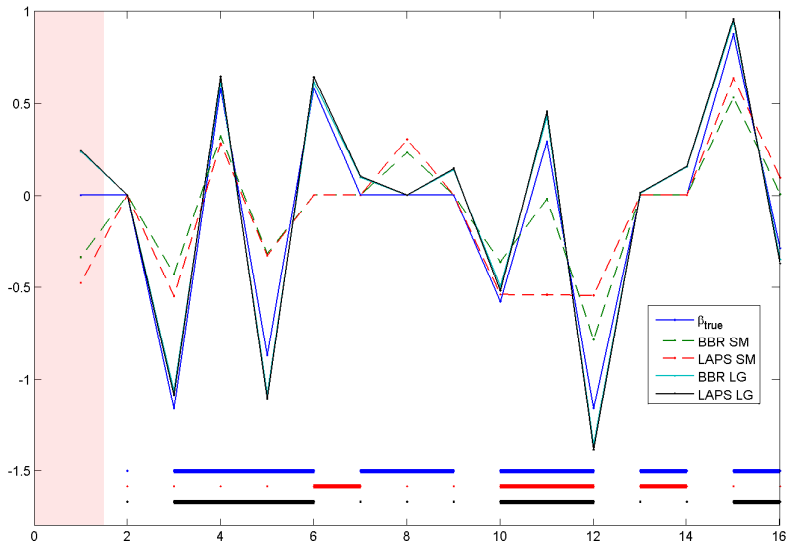


Figure 4.6: Lasso vs. LAPS on the SIM 2 dataset. The plots show the true, Lasso and best LAPS $\boldsymbol{\beta}$s and LAPS $\mathcal{I}$ (the bands at the bottom of each plot). The intended/true run structure for the SIM 2 data is also shown in blue bands.

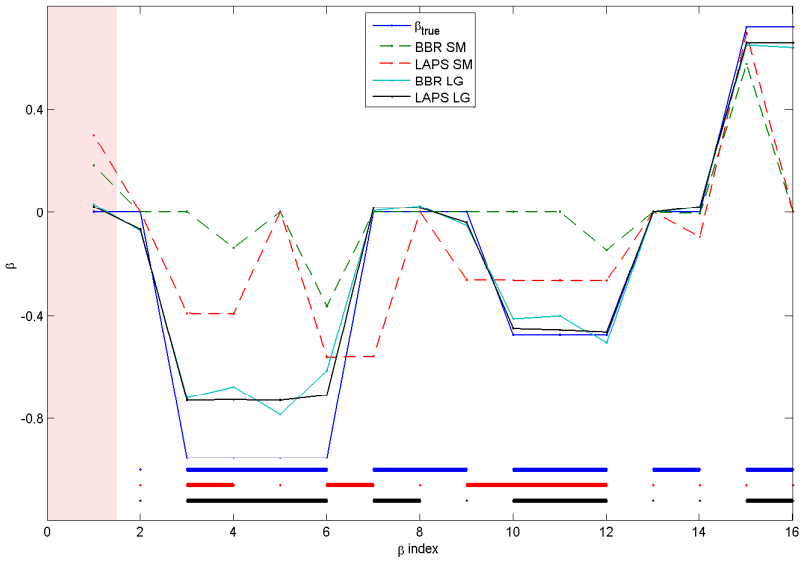


Figure 4.7: Lasso vs. LAPS on the SIM 3 dataset. The plots show the true, Lasso and best LAPS $\boldsymbol{\beta}$s and LAPS $\mathcal{I}$ (the bands at the bottom of each plot). The intended/true run structure for the SIM 3 data is also shown in blue bands.

4.4.2 BF Data

The cylinder, bell, funnel dataset proposed by Saito (1994), is a three class problem, with one input group variable per example[9]. The equations describing the $\mathbf{x}$ for each class have both random noise as well as random start and end points for the generating events of each class, making for quite a lot of variability in the instances. We focus on just two classes, bell vs. funnel. The class labels roughly describe the shape of the examples—bells ramp up and then drop at some point sharply, and funnels spike sharply and then ramp down. As in past studies, we simulated 266 instances of each class to construct the dataset. Figure 4.8 shows the data and a particular instance of each class in bold (bells in red and funnels in blue). Once again, there is only one big group here, consisting the entire set of variable indices (128 predictors).

Both LAPS and Lasso make just one error on one fold in 10-fold cross-validation

[9]This dataset also appears in the next chapter, and readers may benefit from looking at the figure in the CBF section.

predictive accuracy experiments (Table 5.2). Indeed the best LAPS model is very similar to the best Lasso model, with k^*=0, see Figure 4.8. However, some salient properties of the dataset become apparent when examining successive LAPS models with increasing run cohesiveness (i.e., fixing at λ^*, increasing k). As can be seen in the figure, the $k = 0.99$ model seems to imply three specific runs which are discriminative—the first run is not so strong, occurs early (indices around 15—30) and primarily focuses on "early" rising bells from the funnels. The next run is the most significant, occurs right afterwards (from about 35—40) and is the region of the data where the two classes are most segregated[10]. Finally, a short positive run around 50 works in combination with the previous two runs—by this index location, a bell should be on the ascendency compared to a funnel.

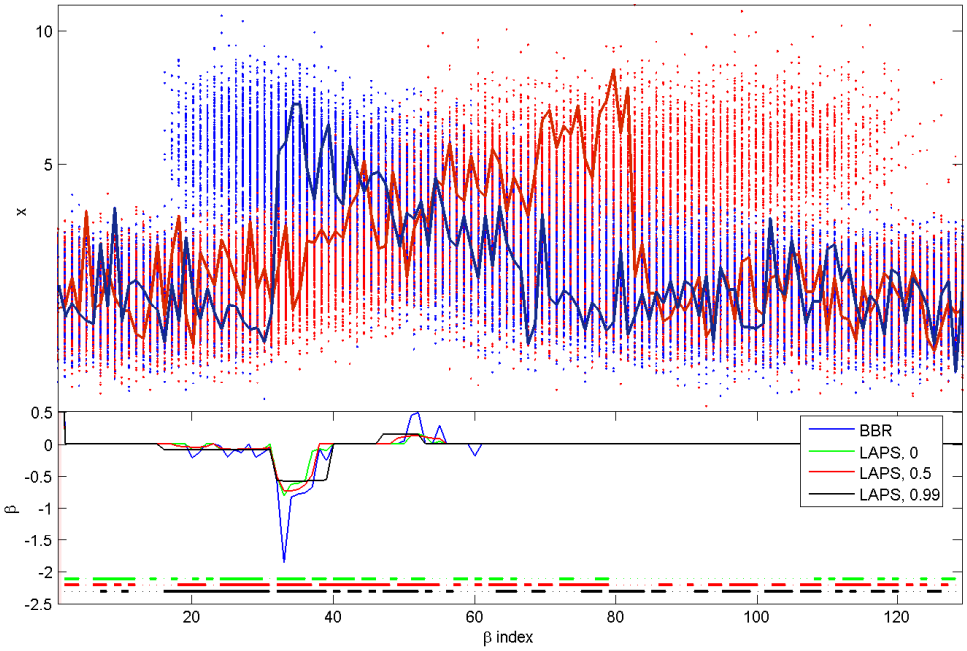


Figure 4.8: Lasso vs. LAPS on the BF dataset. The top portion of this plot shows the whole dataset (no lines connecting the data points for clarity) and two examples. The bottom portion plots the Lasso and best LAPS $\boldsymbol{\beta}$ and corresponding $\mathcal{I}$s. For the other LAPS models in this BF plot, λ is held fixed and k is varied.

[10]While the region around 80—100 also appears to be similarly segregated, a careful look reveals it is actually much more mixed, because some examples for both classes have "fallen" trajectories by this index location.

4.4.3 NHP study

Our final application concerns a vaccine efficacy study. 136 non-human primates (NHPs) were vaccinated, monitored for a year and then "challenged" with anthrax. Of the 136 NHPs, 93 survived the challenge and 43 died. Repeated measurements during the year (and some up to a year after) assessed over a dozen aspects of the putative immune response. These measurements include an include an immunoglobulin G enzyme-linked immunosorbent assay (IgG, see Figures 4.9, 4.10), various interleukin measures (IL2, IL4, IL6), and a so-called "stimulation index" (SI), to name a few, with the number of measurements varying somewhat from animal to animal (See figures 4.11, 4.12 for some examples). The goal of the study is to understand the predictive value of the various assays with respect to survival. The assays thus define the groups, and we search for runs in their time series measurements.

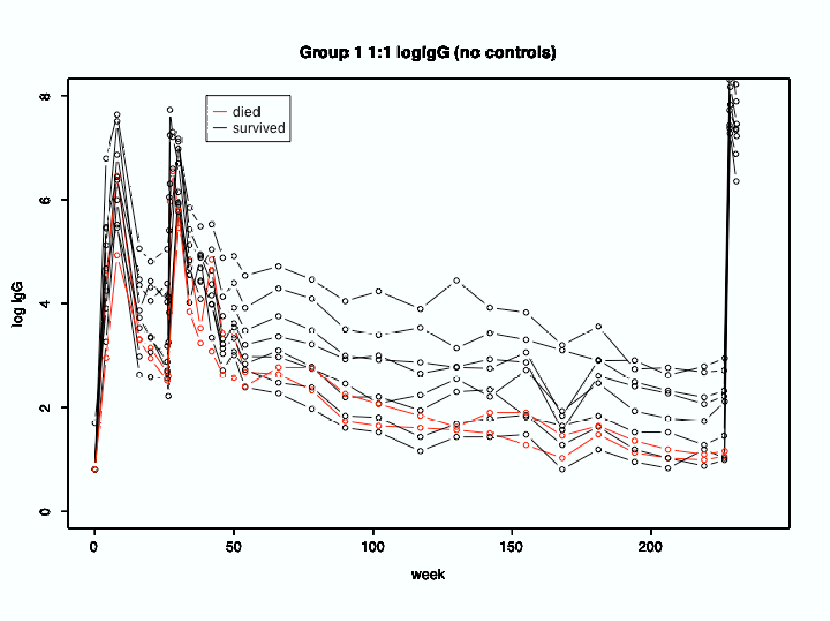


Figure 4.9: IgG assay for a particular dose for all NHPs. In this and plots that follow, red curves are for NHPs that died and black are for those that survived. The x-axis plots time (in weeks), while the y-axis plots the assay measurement. In this plot the two early spikes on the left correspond to booster shots for the vaccine. Notice that low IgG measurements, especially later in the study, seem predictive of death.

The best LAPS model found with $k^* = 0$, looks a lot like the best Lasso model

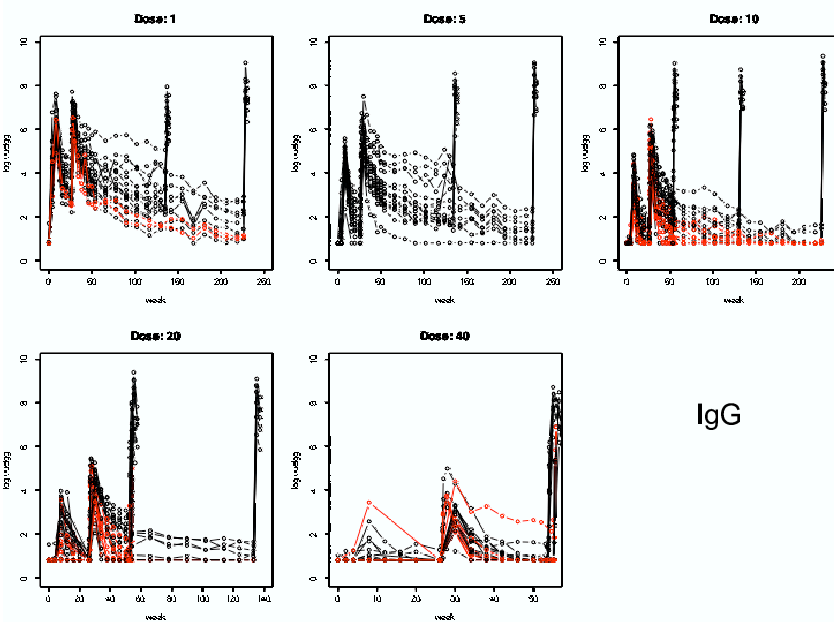


Figure 4.10: IgG assay for all dose levels, for all the NHPs. Again, low measurements seem to be predictive of death.

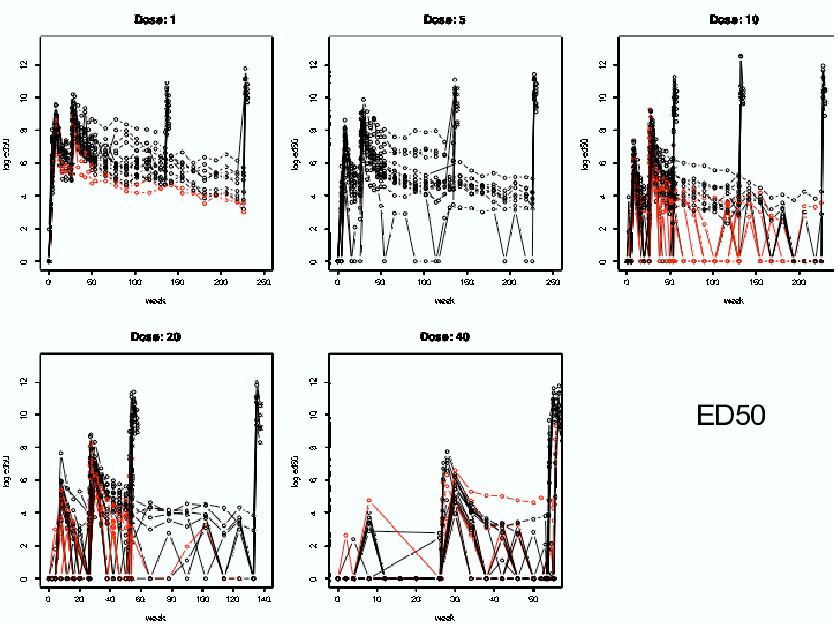


Figure 4.11: ED50 assay for all dose levels, for all the NHPs. This is a tumor neutralizing factor, and again broadly, low measurements seem to be predictive of death.

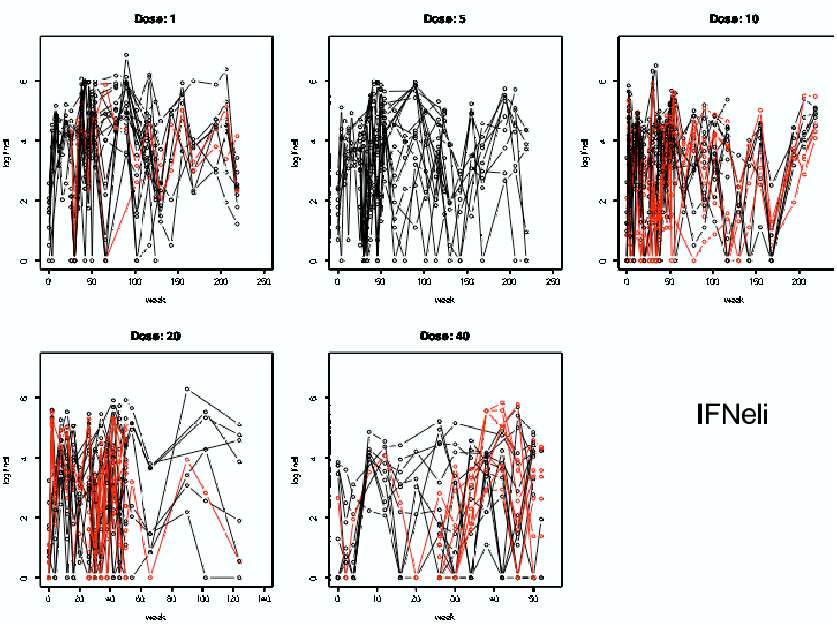


Figure 4.12: IFNeli assay for all dose levels, for all the NHPs. Here, no easily discernable predictive regions exist. A number of assay timeseries feature the two classes in this manner.

found. Again, it is instructive to look at the LAPS models obtained by varying k (Figure 4.13, holding λ^* fixed). The models are biologically reasonable—high amounts of antibodies that neutralize the toxin predict survival (IgG, ED50/TNA). High amounts of interleukins (immune system response/signaling molecules), particularly as time increases, are predictive of death (il4eli, il2m, il4m etc.). Finally, the model also allows one to find runs that are likely not predictive—see for example the first half of the il6m assay, which is identified as a single run across different the k values, and consistently set to zero.

4.5 Discussion

In this chapter we present a model based on the Lasso for finding predictive runs in particular types of structured classification problems. We provide the details of the model, an algorithm for inferring its parameters, and results of application on different types of data. Many extensions of the current work are possible, of which we mention a few. In this work, we have assumed a flat prior over partition space—it would be

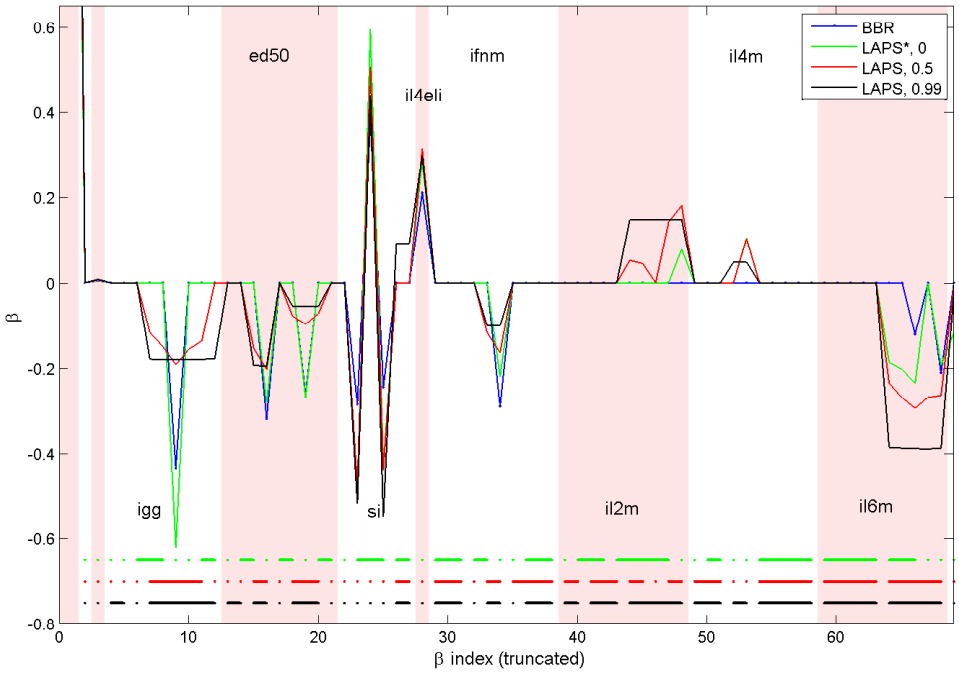


Figure 4.13: Lasso vs. LAPS on the NHP dataset.

Table 4.2: Predictive performance—estimated error rates

Data	**Lasso**		**LAPS**		
	% Err	V^*	% Err	V^*	k^*
SM1	**25.43**	0.45	27.52	0.28	0
SM2	**30.83**	0.15	34.38	0.54	0.99
SM3	35.98	0.15	**30.62**	0.37	0.99
LG1	22.31	0.15	**22.09**	0.54	0.74
LG2	21.14	0.5	**21.09**	0.63	0
LG3	21.86	0.35	21.68	0.19	0.99
BF	0.1887 ± 0.6	200	0.1887 ± 0.6	0.45	0
NHP	30.81 ± 11.97	0.2	**28.02** ± 10.27	0.46	0

interesting to see the effect of various priors on run-partition space. In particular, a non-parametric prior like the Chinese restaurant process prior (adapted for the ordering of variables), might provide for an alternative way for the data to "decide" on the number and type of runs. One could also look at alternatives to the MAP style Bayesian analysis carried out here—numerical simulation may be used to generate a posterior distribution on LAPS model outputs. Finally, extending this model to larger problems (in the number of attributes, the number of training examples and also dimension of structured data—2-D images, for example) raises interesting computational and methodological issues.

4.A The LAPS prior

Viewed as a Bayesian prior, the LAPS regularization term corresponds to a product of multivariate power exponential (MVPE) distributions with power 0.5 Lindsey (1999). The particular MVPE distribution we use has the following density function (for a particular run and with mean zero):

$$f(\boldsymbol{\beta}_{\mathcal{I}_j}|\mu = \mathbf{z}, \Sigma_j) = \frac{c(|\mathcal{I}_j|)}{|\Sigma_j|^{1/2}} \exp -\frac{1}{2}\left[\boldsymbol{\beta}_{\mathcal{I}_j}^T \Sigma_j^{-1} \boldsymbol{\beta}_{\mathcal{I}_j}\right]^{0.5},$$

where the normalizing constant contains $c(|\mathcal{I}_j|) = \frac{|\mathcal{I}_j|\Gamma(|\mathcal{I}_j|/2)}{\pi^{|\mathcal{I}_j|/2}\Gamma(1+|\mathcal{I}_j|)2^{1+|\mathcal{I}_j|}}$. The covariance matrix of this distribution is given by: $\text{cov}(\boldsymbol{\beta}_{\mathcal{I}_j}) = \frac{4\Gamma(nj+2)}{nj\Gamma(nj)}\Sigma_j = 4(n+1)\Sigma_j,\ \forall |\mathcal{I}_j| \in \mathbb{Z}$ ($|\mathcal{I}_j|$ denotes the size/cardinality of $\mathcal{I}_j$). For LAPS, we set $\Sigma_j^{-1} = 2(nj+1)\lambda^2 K_j$. This results in $\text{cov}(\boldsymbol{\beta}_{\mathcal{I}_j}) = 2K_j^{-1}/\lambda^2$. Since the K_j matrices have unit diagonals, the marginal prior variance of every parameter is identical (and equal to $2/\lambda^2$, which is exactly the Lasso model prior variance). This Σ_j setting then results in $s_j = \sqrt{0.5(|\mathcal{I}_j|+1)}$.

Also, the K_j being tri-diagonal results in approximate structural conditional independence[11]. This can be seen by examining an approximating Gaussian graphical model, which would be found by matching the first and second moments (exactly as above). The structural zeros in the approximating Gaussian's inverse covariance result

[11] True conditional independence is not possible for any non-diagonal inverse covariance due to the 0.5 power in the exponent.

in runs being a linear chain graphical model.

4.B Core LAPS problem optimality criteria

There are two distinct optimality cases to check run-wise for a purported solution $\boldsymbol{\beta}^*$ of Equation 4.2 ($\mathcal{I}$ is given): One, if the run is set to zero $\beta^*_{\mathcal{I}_j} = 0$, and two, if all the elements in the run are non-zero $\beta^*_{\mathcal{I}_j} \neq 0$. If $\beta^*_{\mathcal{I}_j} \neq 0$ for the run, the optimality conditions are derived by simply setting the gradient of the objective function to zero.

If $\beta^*_{\mathcal{I}_j} = 0$, we need convex non-smooth analysis results Rockafellar (1970) because the regularization term is non-differentiable at zero. We will use both the notions of the subgradient (a tangent plane supporting a convex function, f. Precisely, the subgradient $\xi \in \mathbb{R}^{|z|}$, of f at z_0 is defined to be any vector satisfying: $f(z) \geq f(z_0) + \xi^T(z - z_0))$ and the subdifferential (∂f which is the set of all subgradients, ξ, at a point. This collapses to the ordinary derivative when the function is differentiable.). We will also use the following theorem: *$\hat{\beta}$ is a global minimizer of a convex function $f(\beta)$ if and only if* $0 \in \partial f(\hat{\beta})$.

Now for $\beta^*_{\mathcal{I}_j} = 0$, use the theorem above. For the matrix norm part of the objective function, the subgradient $\xi \in \mathbb{R}^{|\mathcal{I}_j|}$ satisfies $\left\|\beta_{\mathcal{I}_j}\right\|_{K_j} \geq \xi^T \beta_{\mathcal{I}_j}$ (by the definition). Now, consider the (unique) Cholesky decomposition of the positive definite matrix $K_j = R_j^T R_j$ (also define $\alpha_j = R_j \beta_{\mathcal{I}_j}$). Rewriting the previous condition, we can express the subdifferential as the set ξ_j: $(\beta^T_{\mathcal{I}_j} K_j \beta_{\mathcal{I}_j})^{0.5} = (\alpha_j^T \alpha_j)^{0.5} = \|\alpha_j\|_2 \geq \xi_j^T \beta_{\mathcal{I}_j} = \xi^T R_j^{-1} \alpha_j$. This inequality in turn, holds whenever $\left\|(R_j^{-1})^T \xi_j\right\|_2 \leq 1$, which can also be seen to be equivalent to $\|\xi_j\|_{K_j^{-1}} \leq 1$ (because $K_j^{-1} = (R_j^T R_j)^{-1} = R_j^{-1}(R_j^T)^{-1}$). The theorem then requires that $0 \in \nabla nll(\boldsymbol{\beta})_{\mathcal{I}_j} + \lambda s_j \{\xi_j : \|\xi_j\|_{K_j^{-1}} \leq 1\}$ be satisfied. This finally yields: $\left\|\nabla nll(\boldsymbol{\beta})_{\mathcal{I}_j}\right\|_{K_j^{-1}} \leq \lambda s_j \quad \forall \boldsymbol{\beta}_{\mathcal{I}_j} = 0$. Thus the optimality criteria result:

$$\begin{aligned} \nabla nll(\boldsymbol{\beta})_{\mathcal{I}_j} + \lambda s_j \frac{K_j \boldsymbol{\beta}_{\mathcal{I}_j}}{\|\boldsymbol{\beta}_{\mathcal{I}_j}\|_{K_j}} = 0 \quad \forall \boldsymbol{\beta}_{\mathcal{I}_j} \neq 0, \\ \left\|\nabla nll(\boldsymbol{\beta})_{\mathcal{I}_j}\right\|_{K_j^{-1}} \leq \lambda s_j \quad \forall \boldsymbol{\beta}_{\mathcal{I}_j} = 0. \end{aligned} \tag{4.3}$$

4.C The Approximate Marginal Data Likelihood Score

The S score we use, is based on a Laplace approximation to the posterior distribution. As anticipated, the non-differentiability of the regularization term at zero complicates evaluating it. We use an approximation suggested in Shimamura et al. (2006), which essentially ignores the contribution to the curvature of the posterior by the $\beta^*_{\mathcal{I}^*_j} = 0$ components (in other words, performs a Laplace approximation by only considering the non-zero coefficients/variables—we denote these by the summation till J^*, instead of J for all the attributes). This rather drastic appearing assumption and a straightforward second order Taylor expansion of the negative log posterior results in the score we use. We note here that the assumption is really not as bad as it appears on first glance. Indeed, the posterior is clearly less curved along zero coefficient axes—this can be seen from the optimality conditions, Equation 4.3, by looking at the magnitude of the subdifferential in both cases. Further, simulation studies also show this approximation to be quite reasonable in practice.

4.D Heuristic for the initial $\mathcal{I}$

Given k and λ the procedure (for a single group) is: 1. For each attribute, fit a group Lasso model to the outputs with the current neighbors as the only predictors (a list initially containing just the attribute itself). If the minimum $g_{\lambda,\mathcal{I},k}$ value (restricted to only the attributes in the list) of an expanded neighborhood is better than that for the old neighborhood, update the neighborhood. Otherwise if all expansions result in poorer g value, stop and record the final neighborhood for that variable. 2. Once the neighborhoods for all the variables (in the group) have been obtained, the initial run partition is given by the set of maximal cliques (the largest subgroups where each variable in the clique votes to have the other variable in it's neighborhood, and vice-versa). In our experiments, this heuristic performs quite well at very reasonable computational cost.

References

G. Consonni and P. Veronese. A bayesian method for combining results from several binomial experiments. *Journal of the American Statistical Association*, 90:935 – 944, 1995.

A. Genkin, D. D. Lewis, and D. Madigan. Large-scale bayesian logisitic regression for text categorization., 2004. URL `http://www.stat.rutgers.edu/ madigan/PAPERS/`.

J. K. Lindsey. Multivariate elliptically contoured distributions for repeated measurements. *Biometrics*, 55(4):1277 – 1280, 1999.

L. Meier, van de Geer, S, Bühlmann, and P. The group lasso for logistic regression. Technical report, ETH, Zurich, Switzerland, 2006.

R. T. Rockafellar. *Convex Analysis*. Princeton University Press, Princeton, N.J, 1970.

N. Saito. *Local feature extraction and its application using a library of bases*. PhD thesis, Yale University, 1994.

T. Shimamura, H. Minami, and M. Mizuta. Regularization parameter selection in the group lasso. In *COMPSTAT*, Capri, Italy, 2006.

R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society*, 67(1):91 – 108, 2005.

R. J. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58(1):267–288, 1996.

M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society*, 68(Series B):49 – 67, 2006.

Chapter 5

A Non-parametric Approach to Structured Data Classification[1]

5.1 Introduction

In this chapter, we continue studying the same structured data classification problem of the previous chapter, although from a different viewpoint. In particular, we present an extension to standard decision trees (for example CART, Breiman et al. 1984 or C4.5, Quinlan 1993) that enables them to be applied to classification problems with the structured data as inputs. In so doing, we aim to leverage the interpretability of decision trees as well as their other important benefits like reasonable classification performance and efficient associated learning procedures.

The application that motivated this work is the same as those of the previous chapter. In particular, a similar vaccine efficacy study was a key motivational example. The study in question followed thirty vaccinated NHPs for a year and then "challenged" the animals[2]. Of the 30 NHPs, 21 survived the challenge and 9 died. Repeated measurements during the year assessed over a dozen aspects of the putative immune response. These measurements include an IgG assay, various interleukin measures (IL2, IL4, IL6), and a so-called "stimulation index" (SI), to name a few, with the number of measurements varying somewhat from animal to animal. The goal of the study is again, to understand the predictive value of the various assays with respect to survival.

Our initial approach to this problem used a logistic regression model (Genkin et al., 2004) and treated each assay-timepoint combination as a separate input variable. As

[1]The basic content in this chapter has appeared in "Functional Decision Trees", Balakrishnan S. and Madigan D., ICDM, 2006.

[2]This study was smaller than the study of the previous chapter but otherwise rather similar.

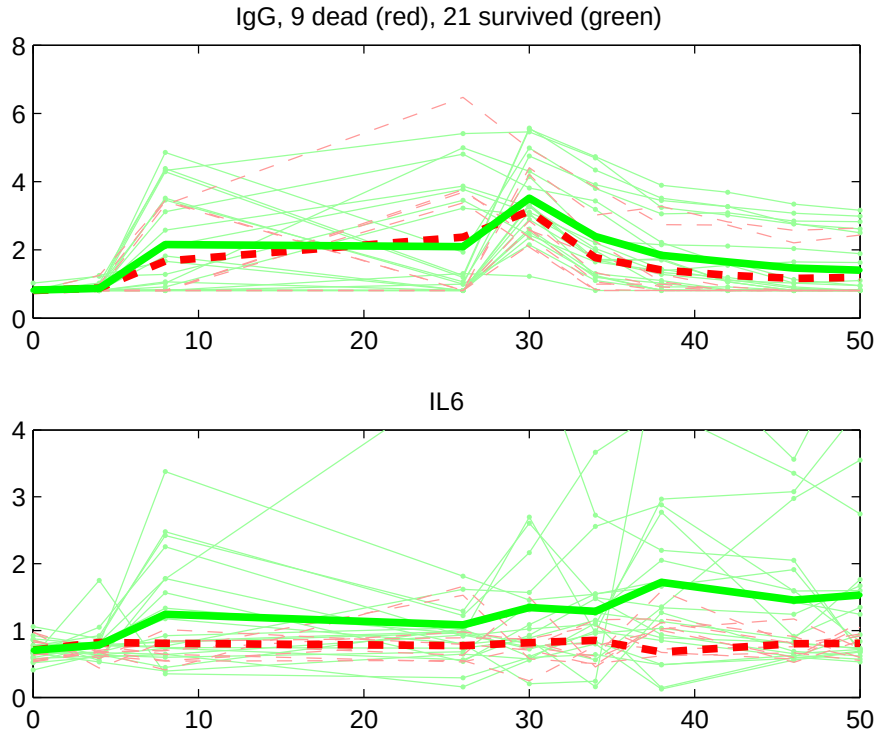


Figure 5.1: IgG and IL6 measurements for all 30 NHPs. Green (solid) represents animals that survived; red (dashed) represents animals that died. The thick curves represent sample means.

mentioned in the previous chapter, while this provided models with good predictive performance, it resulted in models with biologically meaningless set of predictors selected such as IgG at week 46, SI at week 38, and IL6 at week 12. The study immunologists instead sought insights such as "IgG trajectories that rise more rapidly after the 4-week booster shot and fall more slowly after the 26-week booster lead to higher survival probability." In other words, the underlying biology suggests that the shape of the assay curves should be predictive of survival rather than measurements at specific timepoints. Figure 5.1, for example, shows IgG and IL6 trajectories. For IgG, comparison of the thick green curve with the thick red curve shows that higher values are beneficial at the beginning, then lower values, and then higher values again towards the end. For IL6, it appears higher values of the curve in general are beneficial. We shall return to the motivating example in later sections.

More generally, we consider the following multi-class classification problem: the

training dataset comprises n labeled training examples, $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ where $\mathbf{x}_i = [\mathbf{x}_{i1}, \ldots, \mathbf{x}_{id}]$ is a list of d features and $y_i \in \{1, \ldots, c\}$, the c labels. We account for functionally structured input data, by allowing the elements of $\mathbf{x}_i$ to be vectors/lists themselves, each representing an instance of a function of some independent variable. For example, a time series classification problem with only one time-series (or functional) variable of fixed-length T say, as input, would be represented as $\mathbf{x}_i = \mathbf{x}_{i1}$ in our setup, with the single feature $\mathbf{x}_{i1} = [x_{i1}^{(1)}, x_{i1}^{(2)}, \ldots, x_{i1}^{(T)}]$. Here time would be the independent variable.

We allow the inputs to be multivariate—meaning there may be more than one functional variable (i.e., more than one vector/list element of $\mathbf{x}_i$) and also allow for standard (non-functional) discrete/continuous/categorical variables (in this case the relevant components of $\mathbf{x}_i$ will be the corresponding scalars/nominal values). Thus, standard decision trees can be considered a type of special case of the above formulation where all inputs are restricted to length 1.

5.2 Previous Studies

Past approaches to this problem applying standard machine learning algorithms have typically relied on some sort of *ad hoc* and domain specific preprocessing to extract predictive features. A few previous studies look for interesting "events" in the training instances of the functional variables and then construct auxiliary variables based on them. These auxiliary variables are then used by either a particular classifier (decision trees, regression trees and 1-nearest neighbor, Geurts 2001 which retains interpretability), or generic classifiers (Kadous and Sammut, 2005) (with interpretable results available if a rule learning classifier is applied), or used in literals as base classifiers combined via boosting (Gonzalez and Diez, 2004). Another approach is the scheme by Kudo et al. (1999), which constrains the functional variables to pass through certain regions in (discretized) space and disallows other regions. There are also techniques that create features via specialized functional PCA techniques, designed to deal with large data applications (EEG data) where the functional variables are lengthy and numerous. Finally, there are support vector machine based approaches that are typically applied

by defining an appropriate kernel function for the problem domain (Shimodaira et al., 2001).

5.3 Candidate Splits For Functional Variables

In order for a decision tree to be able to process functional variables, we first need to define candidate splits for such variables. In other words, we need to define a procedure that results in a partition of the space of possible function instances (in a manner similar to partitions for discrete/continuous/categorical variables). We describe the idea using the time series example. Consider a binary classification problem, i.e., $y_i \in \{1, 2\}$, and a functional input variable, a time series of fixed length T, $\mathbf{x}_{i1} = [x_{i1}^{(1)}, x_{i1}^{(2)}, \ldots, x_{i1}^{(T)}]$. While many splitting rules can be imagined, we propose the following: consider two representative curves, $\mathbf{x}_r$ and $\mathbf{x}_l$ where $\mathbf{x}_r = [x_r^{(1)}, x_r^{(2)}, \ldots, x_r^{(T)}]$ (and $\mathbf{x}_l$ is similarly defined). The (binary) split is defined by the set of function instances that are closer (in terms of some distance) to one representative curve than the other.

Note that this definition allows for the construction of very flexible partitions. Multi-way splits are an immediate extension and are quite simply defined by considering more than two representative curves. The distance function used can be application specific. In our experiments, we primarily focus on binary splits and two kinds of distance measures: Euclidean distance and dynamic time warping (DTW) distance. The use of DTW distance further exemplifies how flexible this kind of split is, enabling function instances of different lengths to be compared.

Classification with such splits in trees proceeds exactly as with standard decision trees, and the input test function instance follows the branch corresponding to the closest representative curve (with leaf nodes as usual holding predicted class labels).

Note that for a given distance measure, different choices for $\mathbf{x}_r$ and $\mathbf{x}_l$ lead to different candidate splits. Each candidate split corresponds to a partition of "function space" into two regions, functions within one region being more similar to each other than to functions in the other region. Our proposed approach rests on two basic assumptions. First, the partition should be interpretable. That is, the choice of $\mathbf{x}_r$ and $\mathbf{x}_l$

and the particular distance measure should lead to sets of functions that correspond to recognizably homogeneous sets of curves. Second, the true classification rule needs to be smooth with respect to the chosen distance measure. That is, functional inputs that are close together according the distance measure should generally belong to the same output class. What we attempt to show in later sections is that straightforward choices for $\mathbf{x}_r$, $\mathbf{x}_l$, and the distance measure lead to functional-split-enabled decision trees with good classification accuracy and tree structures that provide valuable insights.

5.3.1 Finding Representative Curves

Having provided the intuition for defining the candidate test on the basis of proximity of the function instances to representative curves $\mathbf{x}_r$ and $\mathbf{x}_l$ (quantified in some manner), we now focus on how these curves can be automatically obtained from the data.

Note that the regularity assumption we make, is equivalent to assuming that the instances (or curves) cluster in some manner in the input domain. Consequently, a simple idea is to perform a functional variable-specific clustering, and then use cluster representatives in the candidate tests. The choice of different clustering procedures may lead to different decision trees and in general this choice will require some application-specific consideration.

In our experiments we used two clustering procedures: standard k-means clustering ($k = 2$ for binary partitions) with the Euclidean distance function between two instances of the same length, and a clustering procedure using DTW distances between instances. With DTW distance, the mean of the curves is not a particulary well-motivated representative of a cluster (instances look more like warped/time-shifted versions of each other than the mean). We instead use an instance as representative of the cluster—in particular, we perform the following EM-like iterations to find the representatives: a) Set the cluster representative to be the instance which is closest (has smallest combined DTW distance) to all the other instances in the cluster. b) Reassign instances to clusters based on their distance to the representatives[3]. We will refer to these procedures

[3]Note that this procedure bears resemblance to complete-link hierarchical clustering.

as "Euclidean clustering" and "DTW clustering" in the remainder of this chapter.

5.3.2 Choosing Good Splits

While reasonable clustering procedures can provide reasonable representative curves, most standard clustering algorithms are only guaranteed to converge to locally optimal solutions. The standard approach to alleviate this problem is to do multiple restarts (multiplicity m) initialized randomly and pick the tightest clusters found.

Recall that in our application, however, the criterion for "goodness" of a candidate test is not how tight the found clusters are, but rather how well the representative curves partition the data class labels. Typical measures of partition purity used include entropy, information gain and the Gini diversity index. In our experiments we use the Gini index.

Summing up, in order to find good (high purity) splits for a functional variable we perform multiple restarts of clustering with random initializations (setting $m = 1000$, unless noted otherwise) and pick the partition (the representative curves summarize/define this partition) that has highest Gini index. This search procedure can be easily plugged in to standard divide and conquer decision tree building algorithms like C4.5 and CART, and Algorithm 5 provides an outline in high level pseudo-code.

Algorithm 5: Search procedure for functional variable split

Data: subset of the training data D.
Result: Representative curves $\mathbf{x}_r, \mathbf{x}_l$ that partition the input data D_l, D_r ($D = D_l \cup D_r$), *score* of partition.
Initialize $best = [0, \phi, \phi]$ (stores $[score, D_l, D_r]$).
$\mathbf{x}_r, \mathbf{x}_l = \phi$.
for $j = 1, 2, \ldots, m$ **do**
    Run clustering procedure with random initialization. Obtain candidate representative curves and partition.
    Compute *score* (e.g., Gini index using candidate partition).
    if *score is better than current best score* **then**
        Update $best, \mathbf{x}_r, \mathbf{x}_l$.
    else
        Continue.
    end
end

5.4 Applications

We now describe applications of our algorithm to both simulated and real datasets (see Table 5.1 for details). We will first examine three simulated datasets, the cylinder, bell, funnel dataset (CBF), an extension of it we created (CBF-2), and the control chart (CC) dataset. Table 5.2 provides some details about the predictive performance

Table 5.1: Dataset Descriptions

Data	**Src.**	n	d_f	c	T_r	**Protocol**
CBF	*	798	1	3	128	10-fold CV
CBF-2	*	798	2	2	128	10-fold CV
CC	1	600	1	6	60	10-fold CV
JV	1	370	12	9	7-29	test 270
Bone	2	96	1	2	100	leave one out
NHP	*	30	7	2	7-10	leave one out

*: Own/Simulated, 1: UCI KDD Archive (Hettich and Bay, 1999), 2: Ramsay and Silverman (2002). n: Num. of observations, d_f: Num. of functional variables, c: Num. of classes and T_r: length of the functional instances.

experiments. For each dataset, we compare predictive error rates to one or more of the following: 1. **LR** (an L_1 constrained logistic regression classifier), using BBR[4] (Genkin et al., 2004) (for binary classification problems) and BMR[5] (for multi-class problems) trained with 10-fold CV used to pick the hyperparameter from amongst the set $\{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$. This is a reasonably fair baseline that represents state-of-the-art classifier performance (see Genkin et al. 2004, for how BBR compares to SVMs etc.). 2. **Seg.** (segmented inputs), which are the best previously published results from Geurts (2001) on various classifiers that use segmented auxiliary variables as input. 3. **Fnl.** (functional), which are the best previously published results of Geurts (2001) using comparably interpretable classifiers constructed by combining functional patterns and decision trees. 4. **Best**, which are the best published results otherwise known (not any of the other three categories).

For datasets where 10-fold CV was used to estimate error rates (CBF, CBF-2 and

[4] `http://www.stat.rutgers.edu/~madigan/BBR`

[5] `http://www.stat.rutgers.edu/~madigan/BMR`

CC), the functional decision trees (**FDT**) were pruned by training on 8 out the 10 folds, and picking the sub-tree of the full tree that gave smallest error on the 9th fold (the pruning set). Finally, prediction errors were counted on the remaining 10th fold. For leave-one-out protocol datasets, no pruning was done.

Also, in order to display the functional splits we will use the following conventions in displayed trees throughout: a functional split will be displayed by showing the functional variable name, a $<$ symbol, followed by a unique integer for the split. For example, x1 $<$ 1 represents a split on the functional variable 1 ($\mathbf{x}_{i1}$), and the index 1 likely indicates this is the root split. Further, for any split, **the left branch representative** $\mathbf{x}_l$ will be shown in plots by a **solid line** and the **right branch representative** $\mathbf{x}_r$, will be shown by a **dashed line**.

5.4.1 CBF

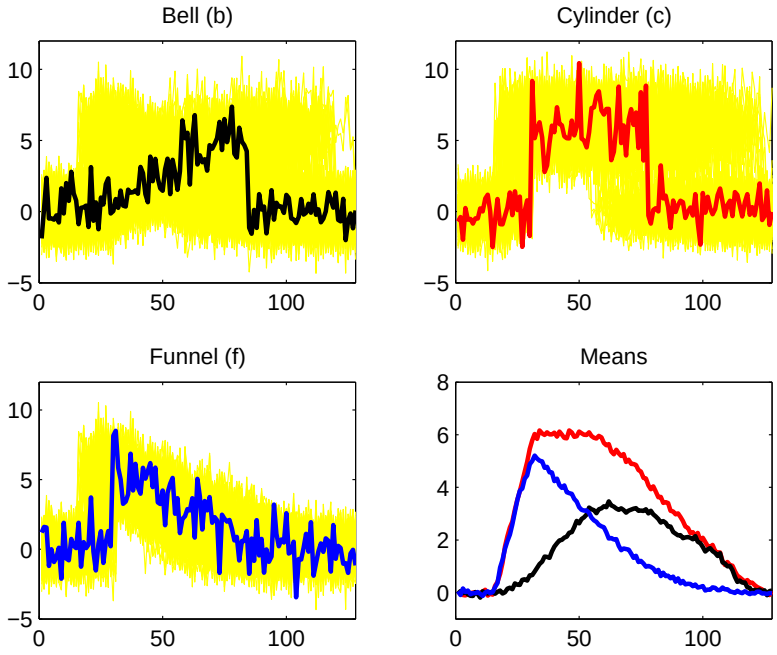


Figure 5.2: Cylinder, bell, funnel dataset.

The cylinder, bell, funnel dataset proposed by Saito (1994) is a three class problem $y_i \in$\{b,c,f\}, with one time series (functional) variable of fixed length. The equations

describing the functional attribute for each class have both random noise as well as random start and end points for the generating events of each class, making for quite a lot of variability in the instances. As in past studies, we simulated 266 instances of each class to construct the dataset. Instances of each class are shown in Figure 5.2, where also shown is a particular instance of each class in bold and the computed class means in the bottom right panel. Although this is essentially an easy classification problem

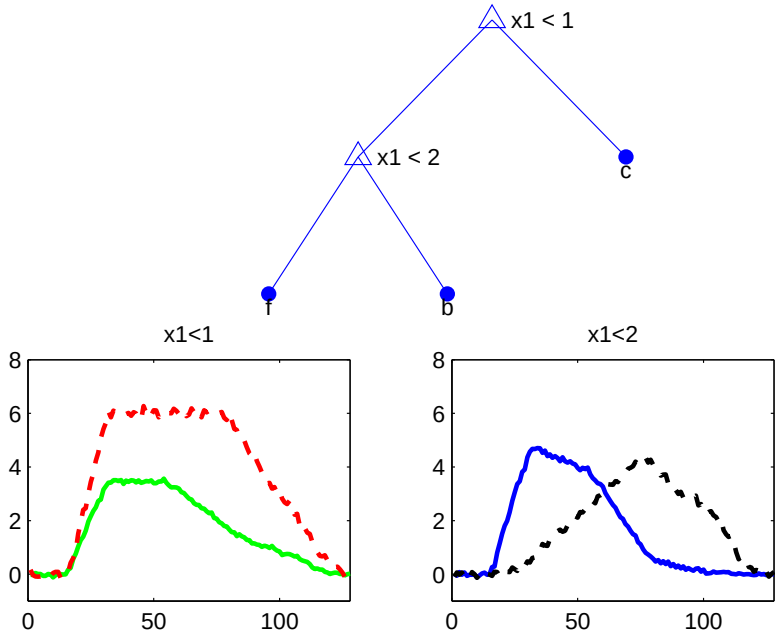


Figure 5.3: CBF results: pruned tree, splits.

(reported accuracies are in the high 90's), it is often used as a sanity check when testing algorithms that perform functional variable classification. As far as predictive performance goes, our procedure also performs at par with some of the best known methods on this dataset (see Table 5.2). The functional decision tree provides a highly interpretable representation. Shown in Figure 5.3 is a pruned decision tree constructed on the whole dataset using Euclidean distance[6]. The leaf splits are, as expected, very representative of the class means (cf., Figure 5.2).

[6]Note that the DTW results are reported in Table 5.2. Euclidean clustering results are slightly worse.

5.4.2 CBF-2

This is an artificial dataset we created that extends the CBF dataset by adding another independent functional variable. This second functional variable we also set to be a cylinder, bell or funnel instance. Finally, we create a binary classification problem with these inputs by assigning patterns to be class 1 if and only if the first variable instance $\mathbf{x}_{i1}$, is a cylinder and the second variable instance $\mathbf{x}_{i2}$, is a bell (again, we simulated 266 instances of each class for each variable). The classification problem is pretty much of the same hardness as the original CBF problem (see the predictive results table). We choose to work with this dataset because it is a perfect example to apply functional decision trees to: a combination of both input functional instances carry the entire predictive signal. The interpretability of the learned tree highlights how effective functional decision trees can be—see Figure 5.4. The splits, one on each variable, correspond exactly to the true data generating mechanism, and this mechanism is evident in the plots of the functional splits.

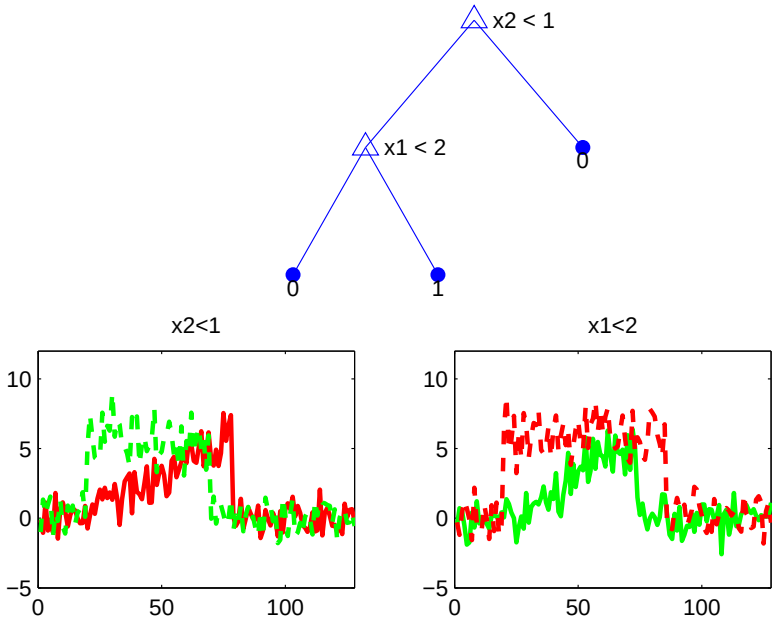


Figure 5.4: CBF-2 results: learnt tree, splits. Branches predicting class 1 are shown in red.

5.4.3 Control Chart

This is an artificial dataset in the UCI KDD Archive[7] consisting 100 objects of each of the six classes. The instances of each class ∈{normal,cyclic,up,down,increasing,decreasing} are defined by 60 time points and the label broadly describes the behavior of the function with time—see Figure 5.5 (Up/Down denote a sudden jump in the series up/down respectively) . Euclidean clustering performs best on this dataset, with competitive

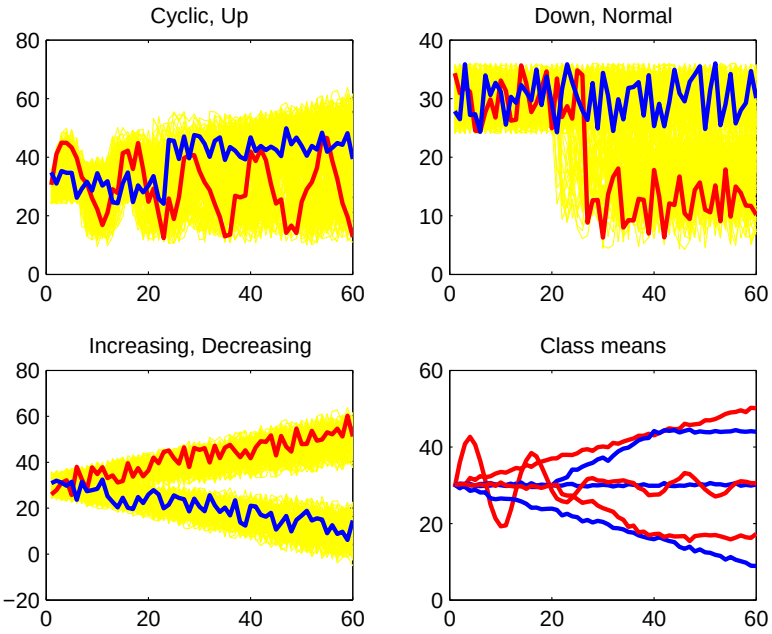


Figure 5.5: Control chart dataset.

predictive accuracy to other techniques (Table 5.2). Figure 5.6 shows the complete functional decision tree along with select functional splits. Notice that the highest level split (x1 $<$ 1) corresponds broadly to partitioning the instances as those that generally increase and those that generally decrease (cyclic and normal are arbitrarily assigned to one or other). Appealingly, other internal splits display similar sorts of intuitive behavior (see x1 $<$ 4, for example) and leaf splits display strong representative instances of the component classes (see x1 $<$ 9 and x1 $<$ 10).

[7]Hettich and Bay (1999), `http://kdd.ics.uci.edu`

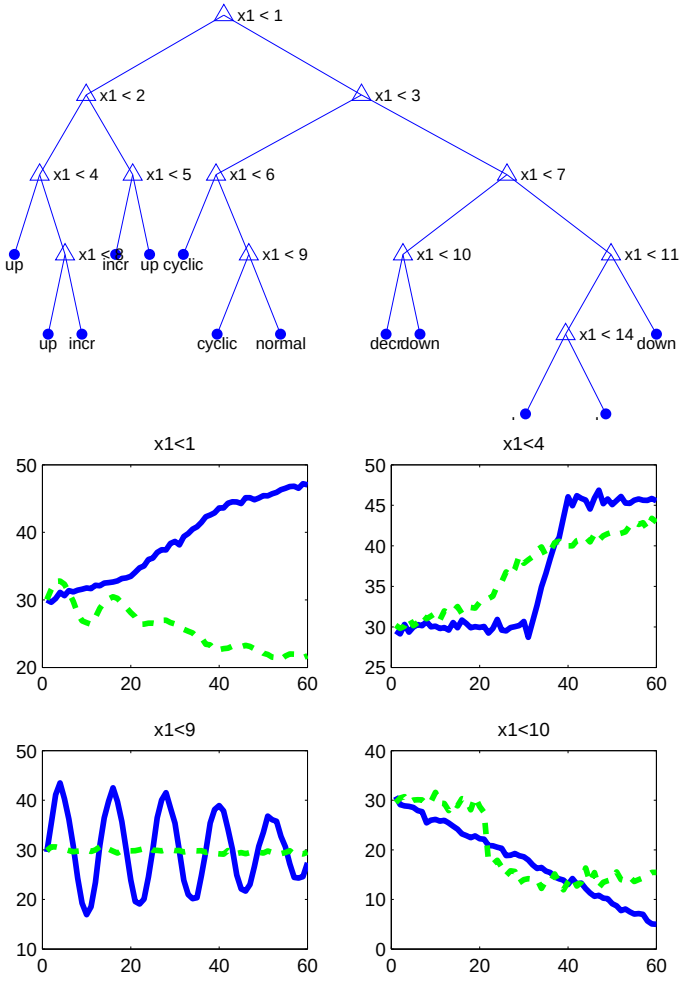


Figure 5.6: Control chart: learnt tree and some splits.

5.4.4 Japanese Vowels

Another dataset in the UCI KDD Archive, the Japanese Vowels dataset was first used in Kudo et al. (1999). The classification problem is a speaker identification task and the dataset consists the utterance of the Japanese vowels "a" and "e" by nine male speakers ($y_i \in \{1, \ldots, 9\}$). Each utterance is described by 12 temporal attributes, which are 12 time-varying LPC spectrum coefficients (for details of how these attributes were obtained see Kudo et al. 1999). An important challenge this dataset poses to many standard classification techniques is that the input sequences are of variable length (owing to the variable length of each utterance). For this reason, our results are

obtained by applying the DTW clustering procedure, for which variable lengths are not a problem.

For this problem, since the number of classes is fairly large compared to the number of examples of each class, we constructed a 15-bit error correcting output code classifier (Dietterich and Bakiri, 1995) from the functional decision trees. Although our prediction accuracy isn't quite at par with the best methods for this problem (see Table 5.2) it is better than that of the other functional classifier that produces interpretable outputs, Geurts (2001). More interesting for this dataset, are the learnt decision trees and their comparison with Kudo et al.'s analysis of the problem. Shown in Figure 5.7 is the tree learnt for speaker 1 (a 1-vs.-all binary classification problem), select splits from the tree, and previously published results based on Kudo et al.'s analysis (who also construct individual speaker classifiers). Kudo et al.'s procedure constructs regions in space through which the speakers curves mostly should (solid black rectangles) and mostly should not (dashed purple rectangles) pass. The qualitative correspondence between these regions and the functional splits we obtain is quite striking.

5.4.5 Bone

This second real dataset records the planar cross-sectional outline of the intercondylar notch from the knee joint (on the femur) of adults. The data has been collected from exhumed skeletons and includes concomitant information such as the sex of the individual. We obtained the data from Ramsay and Silverman's book (Ramsay and Silverman, 2002) data section[8]. The raw data (outline of the coordinates of the bone) is first parameterized by arc length and sampled—see Ramsay and Silverman (2002) for details of this standard pre-processing. This arc length parameterized data is what we set (both x and y coordinates) as our single two-dimensional functional variable (the independent variable being arc length). The binary classification problem involves predicting arthritis and past analyses of this problem have shown the notch shape to be an important predictor. Besides obtaining competitive predictive accuracy on this dataset,

[8] `http://www.stats.ox.ac.uk/~silverma/fdacasebook/`

the learnt tree displays interesting splits. Firstly, the tree contains both non-functional splits (on sex of the individual) and functional splits. Second, the functional splits themselves are instructive about bone shape and arthritis implications—see Figure 5.8 (shown with a comparable plot from the literature). Corroborating the conclusions James and other authors reached by independent analyzes on the same dataset (James and Silverman, 2005; Ramsay and Silverman, 2002), the functional splits we learn show that both variability in the y-direction (shrunken bones) and bending to the right are predictive of a greater risk of arthritis.

5.4.6 NHP study

As described previously, this dataset involves time series measurements (of different lengths) related to the state of the immune system of vaccinated NHPs. Once again, the tree we learn is competitive in predictive accuracy. Figure 5.9 shows the learnt tree and functional splits (as in the introduction, red corresponds to branches for predicting death $y_i = \text{d}$, while green corresponds to NHPs that survive, $y_i = \text{a}$). The splits are biologically sensible. A increasing IL6 trajectory is predictive of survival; A rapid early rise in SI is predictive of death; an early rise in TNF (tumor necrosis factor) is predictive of survival[9].

Table 5.2: Predictive Performance—Error Rates

Data	**Best**	**LR**	**Seg.**	**Fnl.**	**FDT**
CBF	0^1	2.77	0.5	1.17	0.13^D
CBF2	-	5.29	-	-	0.25^D
CC	0.33^2	10.33	0.5	2.33	2.0^E
JV	3.8^3	-	3.51	19.4	9.46^D
Bone	-	21.86	-	-	19.79^E
NHP	-	33.33	-	-	26.67^E

1: Kadous and Sammut (2005) 2: Geurts and Wehenkel (2005) 3: Kudo et al. (1999)
D: DTW clustering, E: Euclidean clustering.

[9]The results also broadly agree with qualitative results obtained from LAPS in the previous chapter. However, we leave the comparison at this abstract level because these are two different datasets.

5.5 Discussion

In this chapter, we presented a simple and effective extension to decision trees that allows them to operate on functional input variables. We presented results showing that these functional decision trees are accurate and produce interpretable classifiers.

Many extensions to the basic idea presented here suggest themselves; we describe a few. The representative curves can be generated by more sophisticated clustering algorithms; of particular interest would be ones designed for clustering functional curves. For example, the one proposed by James and Sugar (2003). Also, a range of algorithms from model-based clustering (e.g. HMM based) to non-parametric clustering (e.g. Gaussian processes based clustering methods) might be applied.

Further, one is not limited to decision trees as the base classifier either. An alternative way to view a single functional split is that it defines an auxiliary variable that may be used in addition to standard features in any classification algorithm. Multi-way splits, for example, might be particularly powerful features in multi-class problems. Finally, predictive accuracy can likely be improved by boosting these functional decision trees, a topic we are currently investigating.

References

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

T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263 – 286, 1995.

A. Genkin, D. D. Lewis, and D. Madigan. Large-scale bayesian logisitic regression for text categorization., 2004. URL `http://www.stat.rutgers.edu/ madigan/PAPERS/`.

P. Geurts. Pattern extraction for time-series classification. In L. de Raedt and A. Siebes, editors, *Proceedings of PKDD 2001, 5th European Conference on Principles of Data Mining and Knowledge Discovery*, LNAI 2168, pages 115 – 127, Freiburg, September 2001. Springer-Verlag. URL `http://www.montefiore.ulg.ac.be/services/stochastic/pubs/2001/Geu01a`.

P. Geurts and L. Wehenkel. Segment and combine approach for non-parametric time-series classification. In *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, October 2005. URL `http://www.montefiore.ulg.ac.be/services/stochastic/pubs/2005/GW05a`.

C. J. A. Gonzalez and J. J. R. Diez. Boosting interval-based literals: Variable length and early classification. In A. Kandel M. Last and H. Bunke, editors, *Data mining in time series databases*. World Scientific, 2004.

S. Hettich and S. D. Bay. The uci kdd archive, 1999. URL `http://kdd.ics.uci.edu`.

G. James and B. Silverman. Functional adaptive model estimation. *Journal of the American Statistical Association*, 100:565 – 576, 2005.

G. James and C. Sugar. Clustering for sparsely sampled functional data. *Journal of the American Statistical Association*, 98:397 – 408, 2003.

M. W. Kadous and C. Sammut. Classification of multivariate time series and structured data using constructive induction. *Machine Learning Journal*, 58:179 – 216, 2005.

M. Kudo, J. Toyama, and M. Shimbo. Multidimensional curve classification using passing-through regions. *Pattern Recognition Letters*, 20(11-13):1103 – 1111, 1999.

J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, California, 1993.

J. O. Ramsay and B. W. Silverman. *Applied Functional Data Analysis: Methods and Case Studies*. Springer-Verlag, New York, 2002.

N. Saito. *Local feature extraction and its application using a library of bases*. PhD thesis, Yale University, 1994.

H. Shimodaira, K. i. Noma, M. Nakai, and S. Sagayama. Dynamic time-alignment kernel in support vector machine. In *Advances in Neural Information Processing Systems*, volume 2, pages 921 – 928. NIPS, 2001.

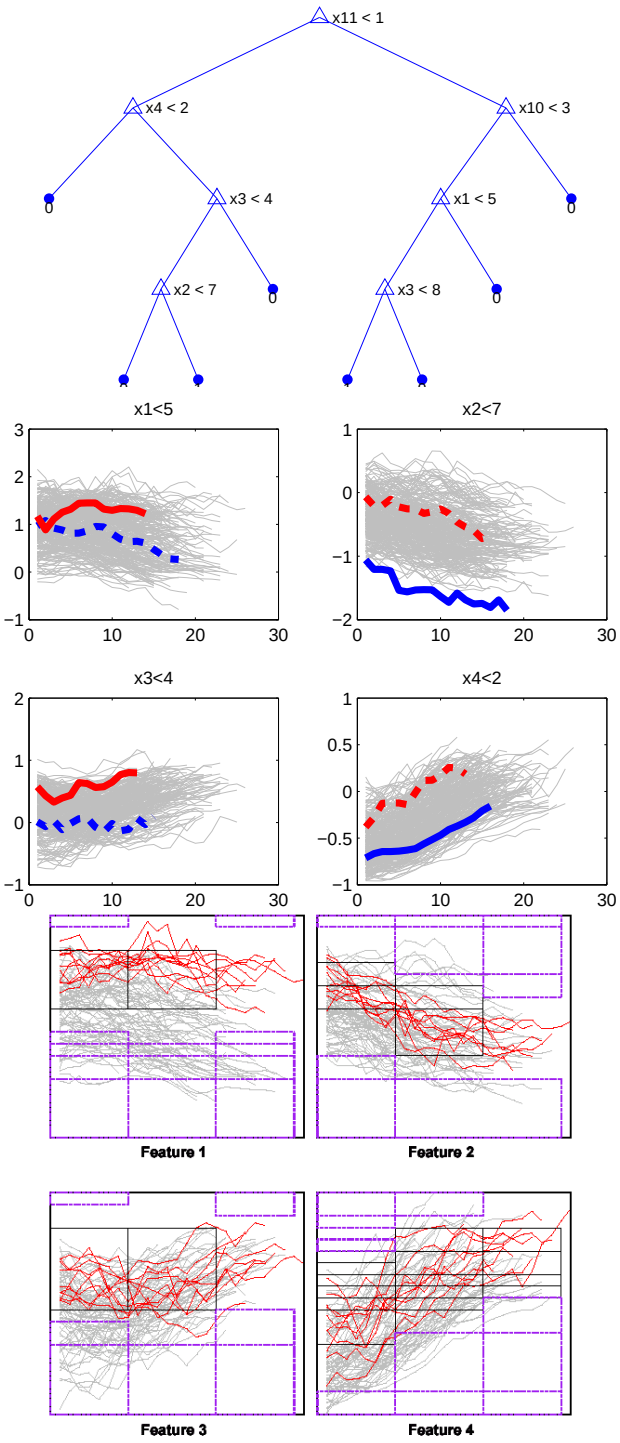


Figure 5.7: Japanese Vowels: Functional splits corresponding to reported results in Kudo et al. (1999). Branches corresponding to speaker 1 have been colored red for ease of comparison. Note: the bottom figure is a capture of a figure in Kudo et al. (1999).

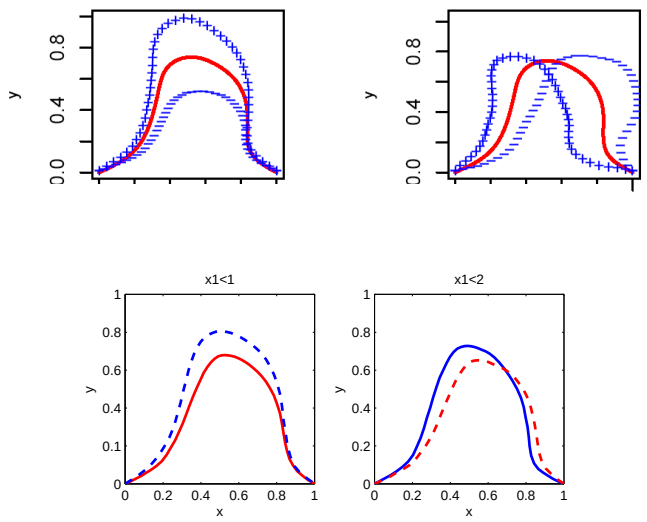


Figure 5.8: Bone data comparative results. Top Row figures: captures from a figure in James and Silverman (2005). Plot shows, in blue, the first two principal components of the predictive model they propose. '-' sign curve being for arthritic bones and the '+' curve being for healthy bone shapes (mean bone shape in red). Lower row figures: root and next level split of learnt FDT (tree not shown). Branches of the functional splits predicting arthritis are colored red.

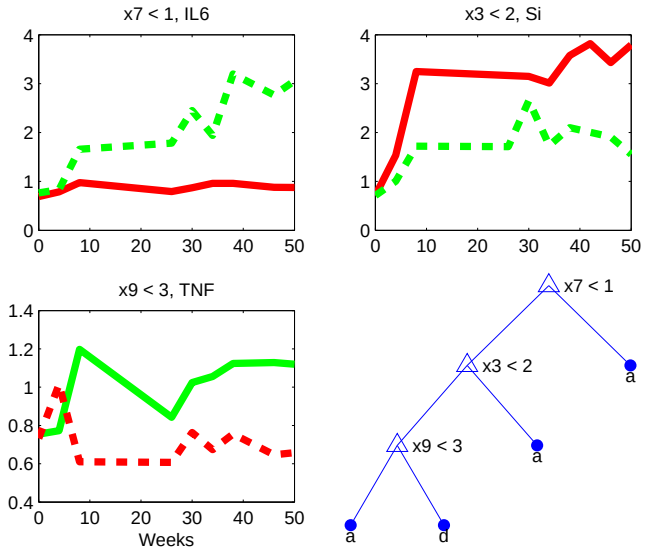


Figure 5.9: NHP learnt tree, functional splits. Branches of the functional splits predicting death are colored red.

Chapter 6

Conclusions

In this thesis we present novel algorithms for supervised binary classification tasks in two non-standard settings— for massive data, where computational limitations arise from being unable to store the dataset in memory, and for structured data problems where the inputs are correlated and interpretable models reflecting this correlation are desired.

Much of the development is motivated by Bayesian theory—which provides a sound theoretical basis to construct algorithms conditioned on a given dataset. We demonstrate that the proposed procedures are all highly competitive to state-of-art classifiers in terms of predictive performance. We propose the following areas of future work:

- Massive Data: The work in Chapters 2 and 3, suggest a number of extensions. In the case of fully Bayesian inference (Chapter 2), for low to medium dimension, a different approach could be the application of efficient numerical integration techniques (for example sparse grid methods, which are essentially modified finite element techniques) using the set of particles as quadrature/knot points. This would yield bounds on the parameter estimates and derived quantities (sparse grid methods enable one to bound integration error). The results of the online algorithm and the MP algorithm in Chapter 3 suggest that a performance-guaranteed bounded memory truly online/streaming approximate L_1 regularized linear model algorithm may be feasible (one pass, using only a pre-specified amount of memory). Indeed, an efficient algorithm for this problem would be useful in a number of applications like: ad placement, collaborative filtering, link analysis, fraud detection etc. The algorithms in Chapter 3 appear to share many commonalities with stochastic gradient based algorithms and their convergence properties may

provide new insights. Also, in order to study the small sample properties of the algorithms in this setting, non-Bayesian frameworks like the online learning model (algorithms like Winnow and exponentiated gradient) could potentially be useful.

- Structured Data: The problem we formalize in Chapters 4 and 5, has large numbers of applications in many diverse practical settings—financial data and medical data to name but two. From the results, it appears that interpretable classifiers can provide a lot insight into the problem domains. Future work may examine the functional decision tree model of Chapter 5 as a generic classifier combination tool. In particular, functional splits applied to kernel space (using kernel k-means for splits, say) would seem to provide powerful hypotheses. For the LAPS work (Chapter 4), a number of questions remain to be resolved, which open up avenues of potential future work. Perhaps the most straightforward issue is improving the the run partition structure search. Another possibility is examining a prior over partition space, and exploring the consequences of this expanded model. The LAPS work also suggests a separate model that may be useful. Here, the data would be generated by a linear chain graphical model (directed or undirected) over the parameters in combination with a suitable sparsity promoting prior (like a spike and slab prior, over latent/hidden indicator variables which would indicate presence or absence of the parameter). Comparisons of this approach and LAPS would be interesting.

Vita

Suhrid Balakrishnan

Education

2002-2007 Ph. D., Computer Science, Rutgers University

1999-2002 M.S., Chemical and Biochemical Engineering, Rutgers University

1995-1999 Bachelor of Technology, Chemical Engineering Indian Institute of Technology, Bombay

Publications

- S. Balakrishnan and D. Madigan, Finding Predictive Runs with LAPS, *ICDM*, to appear, 2007.
- S. Balakrishnan and D. Madigan, Decision Trees for Functional Variables, *ICDM*, 2006.
- S. Balakrishnan and D. Madigan, Algorithms for Sparse Linear Classifiers in the Massive Data Setting, under review, *JMLR*.
- S. Balakrishnan and D. Madigan, A One-Pass Sequential Monte Carlo Method for the Bayesian Analysis of Massive Datasets, *Bayesian Analysis Journal*, **1**(2), 345-362, 2006.