Journal of Machine Learning Research 1 (2007) - Submitted 1/06; Published -

Algorithms for Sparse Linear Classifiers in the Massive Data Setting

Suhrid Balakrishnan SUHRID@CS.RUTGERS.EDU
Department of Computer Science
Rutgers University
Piscataway, NJ 08854, USA

David Madigan MADIGAN@STAT.COLUMBIA.EDU
Department of Statistics
Columbia University
New York, NY 10027, USA

Editor: Peter Bartlett

Abstract

Classifiers favoring sparse solutions, such as support vector machines, relevance vector machines, LASSO-regression based classifiers, etc., provide competitive methods for classification problems in high dimensions. However, current algorithms for training sparse classifiers typically scale quite unfavorably with respect to the number of training examples. This paper proposes online and multi-pass algorithms for training sparse linear classifiers for high dimensional data. These algorithms have computational complexity and memory requirements that make learning on massive datasets feasible. The central idea that makes this possible is a straightforward quadratic approximation to the likelihood function.

Keywords: Laplace Approximation, Expectation Propagation, LASSO

1. Introduction

We 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., 2007; 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; Koh et al., 2007). 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.

This paper presents two basic algorithms for learning L_1-logistic and/or probit regression 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 are sparse, the practical computational cost can be significantly lower.

For massive datasets where t is constant, that is, when given a fixed training dataset, we 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. Hence, this sequential algorithm provides results similar to those of batch algorithms for this problem. The MP algorithm's computational cost is a constant factor higher and memory costs are essentially the same as those of 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.

2. Background and Notation

Throughout this manuscript, we 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 $\mathbf{x}_i = [x_{i1}, \ldots, x_{id}]^T$ 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 work with the posterior distribution of the parameters $\boldsymbol{\beta}$ conditioned on a labeled training dataset D_t, given a prior distribution on the parameters $\boldsymbol{\beta}$:

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

The quantity on the left hand side of (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:

$$\prod_{i=1}^{t} p(y_i|\boldsymbol{\beta}) = \prod_{i=1}^{t} \left(y_i \Phi(\boldsymbol{\beta}^T \mathbf{x}_i) + (1 - y_i)(1 - \Phi(\boldsymbol{\beta}^T \mathbf{x}_i)) \right). \tag{2}$$

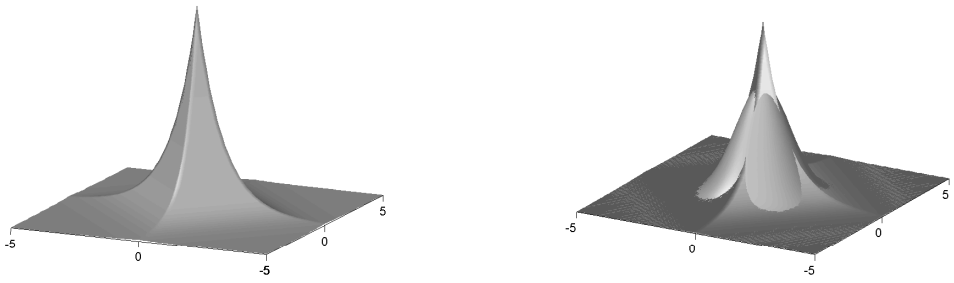
Finding the MAP $\boldsymbol{\beta}$ leads to the optimization problem we wish to solve (now on the log scale):

$$\begin{aligned} & \max_{\boldsymbol{\beta}} \left(\log p(\boldsymbol{\beta}|D_t) \right) \\ \equiv \quad & \max_{\boldsymbol{\beta}} \left(\sum_{i=1}^{t} \log \left(y_i \Phi(\boldsymbol{\beta}^T \mathbf{x}_i) + (1 - y_i)(1 - \Phi(\boldsymbol{\beta}^T \mathbf{x}_i)) \right) - \log p(\boldsymbol{\beta}) \right). \end{aligned} \tag{3}$$

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 (with mean 0):

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

A prior of this form places high probability mass near zero and along individual component axes. It also has heavier tails than a Gaussian distribution—see Figure 1 for plots of the 2-dimensional distributions. It thus favors locations in parameter space with component



(a) (b)

Figure 1: (a) A standard Laplacian distribution, $\gamma = 1$ (b) A superposition of standard (zero mean, unit variance) Gaussian distribution, and the Laplacian distribution 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) presents a convex optimization problem and yields the same solutions as the LASSO (Tibshirani, 1996) and Basis Pursuit (Chen et al.,

1999):

$$\begin{aligned} & \max_{\boldsymbol{\beta}} \left(\log p(\boldsymbol{\beta}|D_t)\right) \\ \equiv \quad & \max_{\boldsymbol{\beta}} \left(\sum_{i=1}^{t} \log \left(y_i \Phi(\boldsymbol{\beta}^T \mathbf{x}_i) + (1-y_i)(1-\Phi(\boldsymbol{\beta}^T \mathbf{x}_i)\right) - \gamma \|\boldsymbol{\beta}\|_1 \right). \end{aligned} \tag{4}$$

The parameter γ in the above problem controls the amount of regularization. Figure 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). Methods such as cross validation can be used to pick its value and algorithms also exist to find solutions for all values of the regularization parameter (commonly called regularization path algorithms). However, we do not address such issues in this manuscript, and we simply assume γ is some fixed, user-specified constant.

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, 2002; Shevade and Keerthi, 2003; Genkin et al., 2007; Koh et al., 2007). Consequently, these algorithms cannot be used in the massive data/online scenario, where memory costs dependent on t represent a significant practical impediment. 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. 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. This requirement in turn, necessitates that we "forget" examples after processing them. We achieve this by maintaining the sufficient statistics of a quadratic approximation in $\boldsymbol{\beta}$ to the log-likelihood of the parameters after incorporating each observation.

We approximate the log-likelihood as:

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

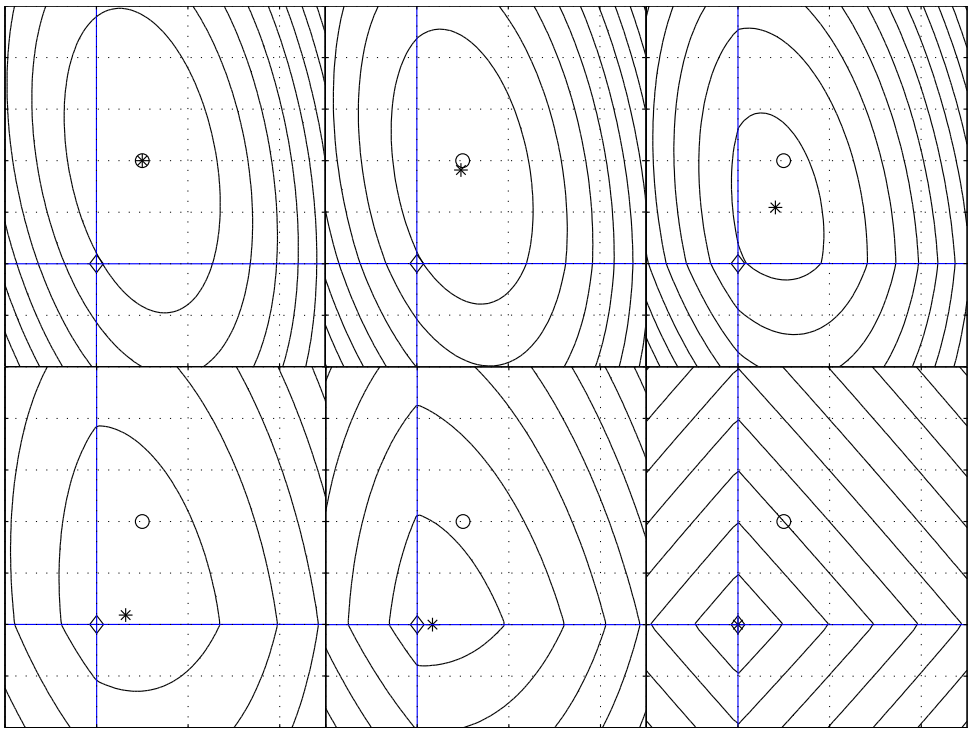


Figure 2: L_1-regularization in two dimensions (i.e., $d = 2$). The axes are the solid lines, the horizontal axis representing β_1 and the vertical axis representing β_2. The diamond represents the origin and the open circle represents the (non-regularized) maximum likelihood solution. The figure shows contours of the function in (4), 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.

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 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 (4) to obtain the modified (approximate) optimization problem:

$$\max_{\boldsymbol{\beta}} \left(\log p(\boldsymbol{\beta}|D_t)\right) \approx \max_{\boldsymbol{\beta}} \left(\boldsymbol{\beta}^T\boldsymbol{\Psi}_t\boldsymbol{\beta} + \boldsymbol{\beta}^T\boldsymbol{\theta}_t - \gamma\|\boldsymbol{\beta}\|_1\right). \tag{5}$$

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:

$$\boldsymbol{\Psi}_{t+1} = \boldsymbol{\Psi}_t + a_{t+1}\mathbf{x}_{t+1}\mathbf{x}_{t+1}^T, \text{ and } \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + b_{t+1}\mathbf{x}_{t+1}. \tag{6}$$

The size of the optimization problem in (5) 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 of 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 manuscript addresses these and other questions. First, we consider how to efficiently obtain the MAP solution of (5), the approximate optimization problem.

3.1 The modified Shooting algorithm

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

$$\max_{\boldsymbol{\beta}} \left(\boldsymbol{\beta}^T\boldsymbol{\Psi}\boldsymbol{\beta} + \boldsymbol{\beta}^T\boldsymbol{\theta} - \gamma\|\boldsymbol{\beta}\|_1\right). \tag{7}$$

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 4, 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., 2007).

In this paper, 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 regression problems (squared loss). Since our approximate optimization problem is also quadratic, the resulting modifications required are straightforward. 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 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 are necessary due to the non-differentiability that the L_1 constraints on $\boldsymbol{\beta}$ result in, see Appendix 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 (7).
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, in this paper 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. Note that $\boldsymbol{\beta}_i$ is the parameter vector at iteration i, which is obtained after cycling through and updating all d components once.

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 B for details). Shooting can be initialized with $\boldsymbol{\beta}_0 = \mathbf{0}$ if no information about the optimal $\boldsymbol{\beta}$ is known or to an appropriate "warm" starting point.

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.

4. An Online algorithm

The quadratic approximation and the Shooting algorithm lead straightforwardly to an online algorithm. After initializing the sketch parameters $\boldsymbol{\Psi}_0, \boldsymbol{\theta}_0$ and the initial parameter vector $\boldsymbol{\beta}_0$, process the dataset one observation at a time. Calculate the quadratic Taylor series 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, producing $\boldsymbol{\beta}_i$ and repeat for the next labelled observation—see Algorithm 2.

Algorithm 2: The Online algorithm.

Data: D_t, γ.
Result: For each i, produces $\boldsymbol{\beta}_i$, an approximation to the MAP estimate of $\boldsymbol{\beta}$ for observations $(\mathbf{x}_1, y_1) \ldots (\mathbf{x}_i, y_i)$.
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 (the data generating mechanism is a logistic regression model with $d = 11$, and $t = 100,000$. For details see the Experiments section of the manuscript). 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, Genkin et al. 2007, publicly available software for batch L_1 penalized logistic regression). See Figure 3, where different colors represent different components of MAP $\boldsymbol{\beta}_i$. Figure 4 shows individual plots of the online and batch estimates for four representative components of MAP $\boldsymbol{\beta}_i$ in blue.

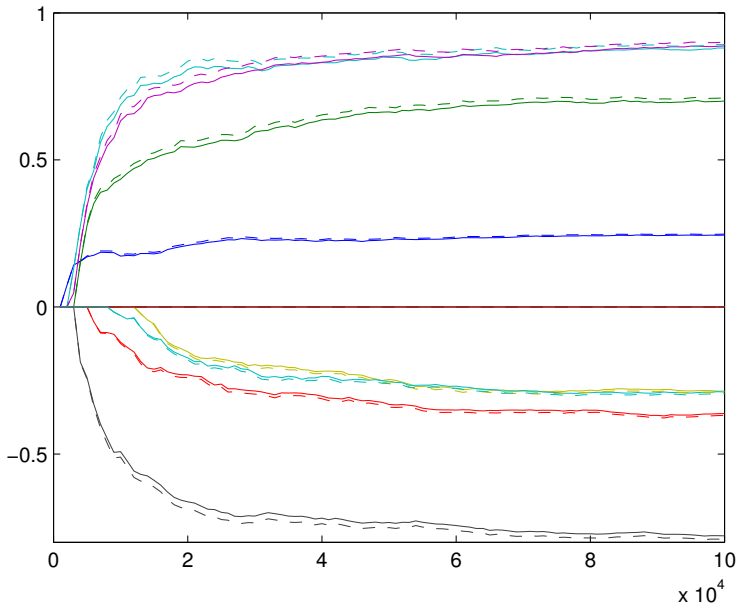


Figure 3: Performance of the online algorithm on a simulated dataset, with regularization parameter $\gamma = 100$ (see text for details). The y-axis is the parameter value, the x-axis the number of observations processed, t.

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 instance, 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 (we remind the reader that the md term, is for the cost of the Shooting algorithm—see 3.1). 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 possible that even very sparse data may result in the full $O(d^2)$ 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 C proves non-divergence of the algorithm in the infinite data limit.

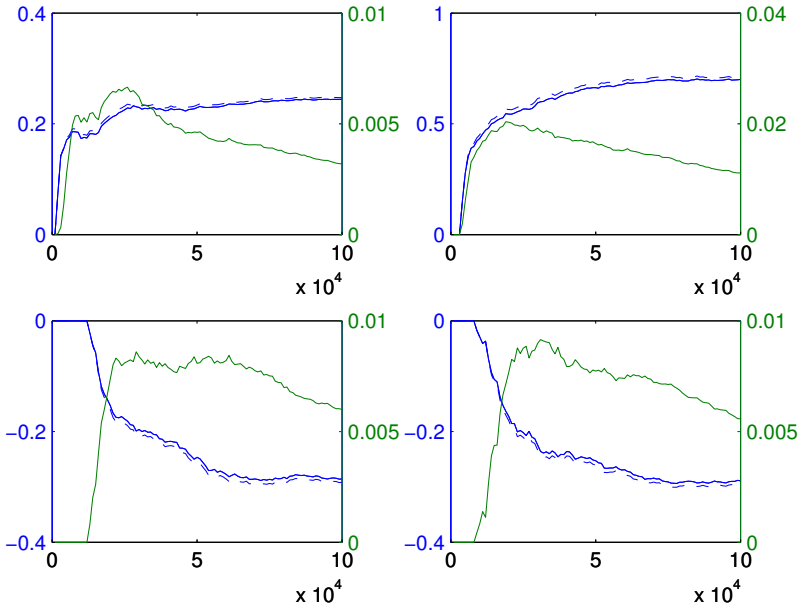


Figure 4: Slightly more detailed version of Figure 3. The panels show four representative parameters from that figure, also showing tapering L_1 loss between the online and batch algorithm estimates on the right axis (in green). Simulated dataset, $\gamma = 100$. Once again, the (left) y-axis is the parameter value and the x-axis the number of observations processed, t.

4.1 Heuristics for improvement/Issues

While one can also obtain parameter estimates for fixed t (batch problems) 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 (4) for some fixed t. Since the online algorithm typically initializes itself far from $\boldsymbol{\beta}_*$, 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, (6), 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 $\gamma\|\boldsymbol{\beta}\|_1$. 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 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

	$t = 2\text{x}10^4$		$t = 6\text{x}10^4$		$t = 10^5$	
$\boldsymbol{\beta}_{true}$	**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 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).

initially to the specified value γ at the end of the dataset[1]). 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 temporarily instead of immediately discarding each observation after processing it. The algorithm then uses the value of the parameter estimates after having seen/processed all the observations in a block to update the sketches for the whole block. Note that this will involve keeping track of the corresponding updates to the sketches for the block (the block's contributions to $\boldsymbol{\Psi}$ and $\boldsymbol{\theta}$). 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 maximizes the approximate one-step look ahead likelihood (although the expressions for this approximation would be slightly different). 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

1. While the choice of this regularization schedule in this setting is understudied in the literature, asymptotic consistency results for a slightly modified form of the problem may be of theoretical interest. We refer readers to Zou 2006, and the references therein.

sequential quadratic approximation that improves the accuracy of estimates when applied to finite datasets and also uses less memory than the online algorithm.

5. A multi-pass algorithm

The block heuristic of the previous section implies that taking *all* term approximations at the final online algorithm MAP $\boldsymbol{\beta}_t$ value would certainly produce better estimates of $\boldsymbol{\Psi}_t, \boldsymbol{\theta}_t$. This in turn would lead to 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.

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 first term is the cost of updating the sketches and the second md term is the cost of the Shooting algorithm).

The worst case memory requirement of the MP algorithm is $O(d^2)$, which is just a constant with respect to t. Expectation Propagation (Minka, 2001b) by contrast requires explicitly storing term approximations and thus has memory costs that scale linearly with t, i.e., $O(t)$. The next subsection presents a modification of the MP algorithm that reduces this worst case memory requirement.

5.1 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 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 $S = \{j : |\Omega_j| \geq \gamma\}$. That is, the active set is the set of variables that are either nonzero and optimal or variables that violate optimality at the start of a pass (the corresponding nonzero elements of the vectors/matrices are denoted by their previous symbols but with a $\tilde{}$ above them). Now, during the pass we keep track of the much smaller matrix $\tilde{\boldsymbol{\Psi}}$, while also keeping track of the unmodified/original full length vectors $\boldsymbol{\theta}$ and $\boldsymbol{\Omega}$. The update for $\boldsymbol{\theta}$ is unchanged and Appendix B shows how to

perform the update for the full length vector $\mathbf{\Omega}$ in small space. The algorithm continues by using $\tilde{\mathbf{\Psi}}, \mathbf{\Omega}$, and $\boldsymbol{\theta}$ from the latest pass to re-estimate the active set, S and so on.

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.

Algorithm 3: The RMMP algorithm.

Data: fixed dataset D_t, γ.
Result: $\boldsymbol{\beta}_z$, the MAP estimate of $\boldsymbol{\beta}$ that solves (4).
Initialize $\boldsymbol{\beta}_0 = \mathbf{0}$, $S = \{\}, z = 1$.
while *not converged* **do**
  Set $\boldsymbol{\theta} = \mathbf{0}$, $\tilde{\mathbf{\Psi}} = \mathbf{0}$, $i = 1$.
  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.
    $\tilde{\mathbf{\Psi}} \leftarrow \tilde{\mathbf{\Psi}} + a_i(\tilde{\mathbf{x}}_i \tilde{\mathbf{x}}_i{}^T)$.
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + b_i \mathbf{x}_i$.
    Update $\mathbf{\Omega}$.
  end
  $\boldsymbol{\beta}_z \leftarrow$ modified Shooting$(\tilde{\mathbf{\Psi}}, \tilde{\boldsymbol{\theta}}, \tilde{\boldsymbol{\beta}}_{z-1}, \gamma)$.
  Obtain new active set $S = \{j : |\Omega_j| \geq \gamma\}$.
  $z \leftarrow z + 1$.
end

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 time requirements for a constant number of passes, are still $O(td^2)$ to do an equivalent batch MAP $\boldsymbol{\beta}$ estimation. Under the same sparsity assumptions as in previous sections, in practice this cost is better quantified as $O(t(k^2 + f^2) + kd)$ (again, the first term is the cost associated with updating the sketches and the second term is the cost of Shooting).

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 fewer 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; the first pass is used simply to determine the size and components of the active set and 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 $\boldsymbol{\Omega}$ implies the practitioner can confirm when convergence to $\boldsymbol{\beta}_*$ has/has not occurred. 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 is bigger than the number of variables in the current active set, for any iteration) in two ways: 1. We retain in the active set variables that were in the active set in the previous iteration 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 $S = \{j : |\Omega_j| \geq \tau\gamma\}$).

In the next section, we place our work in the context of existing literature on similar problems.

6. Related work

Although the Bayesian paradigm facilitates 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 Gaussian distribution (which implies a quadratic approximation of the log posterior 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).

Indeed, 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, 1998). We would like to stress here that many of the above schemes are for the harder task of approximate inference—we are concerned only with the easier problem of approximate convex optimization. The similarities in the approaches are confined to the nature of the approximate (Gaussian) posterior.

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

7. Experiments

We now present examples illustrating the application of the Online, 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., 2007). BBR is publicly available software for Bayesian binary logistic re-

gression that handles the Laplacian prior. We make probit regression comparisons to results obtained using a batch EM algorithm for Laplacian prior based probit regression (we implemented 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. (2007); 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 one intercept term and 10 model coefficients, for a total of 11 fixed parameters. Of the ten model variables, three are intentionally set as redundant variables (set with zero coefficients in the model). The data vectors $\mathbf{x}$, are draws from i.i.d. Gaussian distributions with mean zero and unit variance. For the experiments with the online algorithm (Figure 3, Table 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, "earn", 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 5. We also use just the training portion of RCV1-v2 in some experiments. RCV1-v2 training dataset : $d = 47,236$, $t = 23,149$ (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.

7.1 Results

The low dimensional simulated dataset highlights typical results we obtain with the Online algorithm and the MP algorithm (the RMMP algorithm is not of practical significance in this case). See Table 2. Each column in the table is an 11-dimensional vector which is the MAP $\boldsymbol{\beta}$ estimate of the parameter values (as a reminder, the true parameter values used to generate the data can be seen in Table 1). The parameter estimates from the Online algorithm are quite close to batch estimates, likely due to the relatively large dataset size (t being large relative to d). Also, with very few passes over the dataset, denoted as before by the variable z, we obtain parameter estimates practically identical to those obtained by the batch algorithm. The results in the table are typical for both link functions and over a wide range of settings for the regularization parameter, γ. To show this, the tables report results for both too little regularization ($\gamma = 10$, probit link) and too much regularization

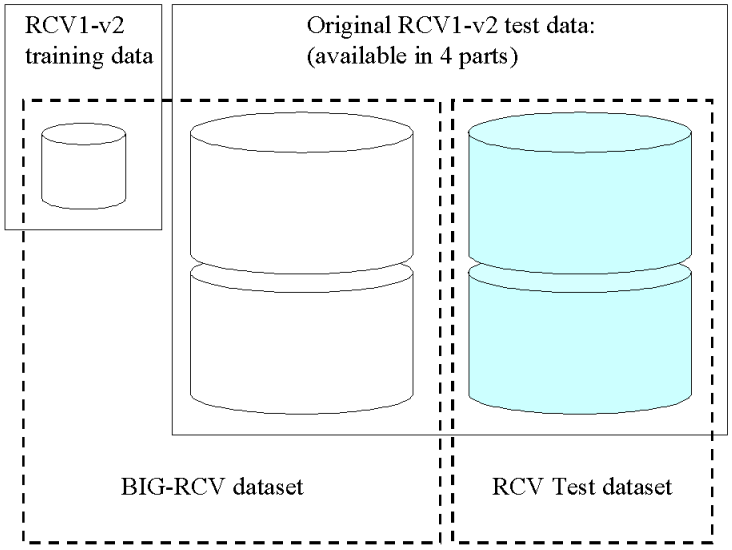


Figure 5: Schematic showing the construction of the various RCV1-v2 based datasets used in the experiments. The solid line bordered rectangles show the data as publicly available, the dashed-line bordered rectangles show the datasets we assembled. The shaded portion of the data is used only during testing.

($\gamma = 100$, logistic link) for this particular dataset. As a guide to assessing convergence in this and other tables that follow, we show the L_1 norm of the difference between the batch algorithm estimates (EM or BBR as appropriate) and the Online, MP or RMMP algorithm iterates (also as appropriate).

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) dataset with $t = 9603$ labelled observations (we use the feature vectors that can be downloaded from the paper's appendix.). 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 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, and presents no memory limitations, we are able to apply the Online and MP algorithms in addition to the RMMP algorithm—see Tables 3, and 4.

For both amounts of regularization the Online parameter estimates aren't particularly good (although between the two settings, the parameter estimates with the higher amount of regularization are better). As discussed in section 5, this is likely due to the relatively high dimensionality compared to the number of examples in the dataset. The MP algorithm improves parameter estimates as expected. For $\gamma = 100$, the MP algorithm converges in

Probit link function, $\gamma = 10$					Logistic link function, $\gamma = 100$				
EM	**Online**	**MP**			**BBR**	**Online**	**MP**		
		$z = 1$	$z = 2$	$z = 3$			$z = 1$	$z = 2$	$z = 3$
0.252	0.250	0.207	0.250	0.252	0.178	0.174	0.168	0.178	0.178
0.764	0.764	0.614	0.755	0.764	0.450	0.435	0.422	0.450	0.450
-0.318	-0.314	-0.263	-0.314	-0.318	-0.124	-0.120	-0.1161	-0.124	-0.124
0.834	0.821	0.667	0.824	0.834	0.713	0.689	0.666	0.712	0.713
0.894	0.880	0.719	0.884	0.894	0.656	0.634	0.613	0.655	0.656
-0.304	-0.297	-0.243	-0.301	-0.304	0	0	0	0	0
-0.782	-0.770	-0.627	-0.773	-0.782	-0.511	-0.493	-0.477	-0.510	-0.511
-0.039	-0.039	-0.037	-0.039	-0.039	0	0	0	0	0
-0.036	-0.036	-0.029	-0.036	-0.036	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
-0.327	-0.322	-0.266	-0.324	-0.327	-0.030	-0.029	-0.028	-0.030	-0.030
L_1 Norm	0.074	0.878	0.050	0		0.0872	0.172	0.003	0

Table 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), the Online algorithm 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 Online/MP algorithm estimates. The results shown here are representative of those obtained for other values of γ as well.

about $z = 6$ iterations to parameter values indistinguishable from BBR—see the left three columns in Table 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 4 for $\gamma = 10$. We point out that this is a huge reduction in the worst case memory required, an approximately 98% reduction ($k = 3000$ 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 ($d = 47,236$, $t = 23,149$), sparsity again enables application of BBR to obtain the batch MAP $\boldsymbol{\beta}$ parameter values, as well as the Online and MP algorithms, although this is quite cumbersome. See Table 5. Again, as expected (examining d vs. t for this dataset), the Online estimates are not very good. The multi-pass algorithms have improved parameter estimates. For $\gamma = 10$ (a fairly high amount of regularization), we find essentially the same qualitative results as the ModApte dataset—it takes about $z = 6$

j	**BBR**	**Online**	**MP**		**RMMP**, $k = 300$		
			$z = 3$	$z = 5$	$z = 3$	$z = 5$	$z = 7$
Intercept	-1.588	-1.404	-1.527	-1.586	-1.451	-1.573	-1.588
9 (bank)	1.188	0.697	0.957	1.185	0.688	1.143	1.188
13 (share)	0.847	0.609	0.793	0.846	0.678	0.839	0.847
147 (acquisit)	0.813	0.562	0.795	0.813	0.696	0.812	0.813
31 (offer)	0.801	0.337	0.618	0.800	0.356	0.772	0.801
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
3 (pct)	-2.264e-2	-2.259e-2	-2.127e-2	-2.240e-2	-3.247e-2	-2.062e-2	-2.264e-2
62 (plan)	-1.757e-2	-1.430e-2	-2.840e-2	-1.779e-2	-3.346e-2	-2.045e-2	-1.757e-2
2 (dlr)	1.552e-2	6.932e-3	1.542e-2	1.548e-2	1.610e-2	1.525e-2	1.552e-2
12 (net)	-1.467e-2	-6.671e-3	-1.956e-2	-1.480e-2	-1.415e-2	-1.643e-2	-1.467e-2
8 (ct)	1.277e-2	3.587e-2	2.870e-2	1.320e-2	2.915e-2	1.776e-2	1.278e-2
L_1 Norm		4.029	1.691	0.034	3.496	0.4027	3e-4

Table 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 $\boldsymbol{\beta}$ (and word stem features they correspond to in brackets), coefficients from BBR, and the Online algorithm, those obtained after a particular number of passes over the data using the MP algorithm (full memory) and parameters from the RMMP algorithm with $k = 300$.

j	**BBR**	**Online**	**RMMP**, $z = 8$				
			$k = 3120^*$	$k = 2000$	$k = 1000$	$k = 600$	$k = 300$
292 (banker)	2.695	1.523	2.695	2.695	2.695	2.695	2.699
20 (4)	2.268	0.617	2.268	2.260	2.260	2.259	2.273
Intercept	-2.010	-1.615	-2.010	-2.009	-2.009	-2.009	-2.005
341 (charg)	1.755	0.832	1.755	1.754	1.754	1.754	1.742
147 (acquisit)	1.572	0.862	1.572	1.572	1.572	1.572	1.568
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
66 (loan)	4.943e-3	9.106e-2	4.944e-3	4.849e-3	4.821e-3	4.849e-3	3.224e-3
134 (agre)	4.488e-3	4.836e-2	4.479e-3	4.720e-3	4.756e-3	4.712e-3	1.677e-2
267 (commerci)	-2.057e-3	0	-2.068e-3	-1.863e-3	-1.897e-3	-1.852e-3	-3.427e-3
28 (stock)	-1.652e-3	-3.879e-2	-1.644e-3	-1.542e-3	-1.560e-3	-1.537e-3	-1.991e-3
56 (interest)	-1.518e-4	-7.623e-2	-1.540e-4	-3.059e-4	-2.983e-4	-3.121e-4	-8.640e-4
L_1 Norm		28.290	1.4e-3	0.047	0.044	0.048	1.269

Table 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 $\boldsymbol{\beta}$ (and word stem features they correspond to in brackets), coefficients from BBR, the Online algorithm, 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**	**Online**	**RMMP,** $k = 1500$			$\boldsymbol{\beta}$ index	**RMMP**
			$z = 2$	$z = 5$	$z = 10$		$z = 10$
12220 (econom)	18.065	16.145	0	18.084	18.065	12220 (econom)	17.234
27407 (moody)	9.909	9.982	7.988	9.904	9.909	37665 (shar)	-11.901
37665 (shar)	-8.201	-3.918	-2.255	-8.118	-8.201	43626 (union)	8.654
46160 (work)	7.144	6.339	4.061	7.133	7.144	27407 (moody)	8.308
5946 (budget)	6.453	6.327	5.142	6.436	6.453	5946 (budget)	8.215
33192 (profit)	-6.211	-3.840	-2.066	-6.159	-6.211	19647 (inflat)	6.326
43626 (union)	6.164	5.789	4.430	6.157	6.164	39539 (statist)	5.782
21160 (july)	5.661	5.093	3.498	5.644	5.661	29641 (obligat)	4.728
19647 (inflat)	5.573	5.437	6.587	5.539	5.573	37471 (sery)	4.621
29641 (obligat)	5.472	6.250	4.810	5.473	5.472	41148 (tax)	4.507
L_1 Norm		24.798	87.940	0.480	0.001		

Table 5: RCV1-v2 results. Left portion RCV1-v2 training dataset, right BIG-RCV dataset.

passes through the dataset to obtain indistinguishable parameter values as BBR (not shown in the table). The RMMP algorithm also gives excellent results in about 10 passes, see the left portion of Table 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 Online and 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—see for example, the right portion of Table 5.

Does training on the entire BIG-RCV dataset actually result in improved predictive performance? To address this, we conducted the following experiment. We obtained the best possible predictive parameters using 10-fold cross-validation on the RCV1-v2 training dataset with a batch algorithm. This is an expensive computation, involving many repeated BBR runs for different values of the regularization parameter (we searched over $\gamma = 0.01, 0.1, 1, 10, 100$). 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 $\boldsymbol{\beta}$. 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 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.

8. Conclusions

In this paper we presented an asymptotically convergent online algorithm that builds sparse generalized linear models for massive datasets. We also presented efficient multi-pass algo-

	"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 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.

rithms that examine observations sequentially and thus enable learning on massive datasets. Both algorithms exploit sparsity of input data. We applied the algorithms to 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.

Acknowledgments

National Science Foundation grants IIS-9988642 and DMS-0505599 and the Multidisciplinary Research Program of the Department of Defense (MURI N00014-00-1-0637) supported this work. We are very grateful for to David D. Lewis for detailed and insightful comments on an earlier draft of this paper.

Appendix A.

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.

Appendix B.

In this appendix we derive the modified Shooting algorithm, Algorithm 1 and discuss its efficient implementation. We derive Shooting by analyzing the subdifferential of the system (Rockafellar, 1970). We need convex non-smooth analysis results because the regularization term is non-differentiable at zero. Reviewing concepts very briefly, the subgradient $\xi \in \mathbb{R}^{|x|}$, of a convex function f at x_0 is defined to be any vector satisfying:

$$f(x) \geq f(x_0) + \xi^T(x - x_0).$$

In words, any vector ξ, such that a plane through $(x, f(x))$ with slope ξ contains f in its upper half-space qualifies as a subgradient (equivalently, a tangent plane supporting the convex function f). The subdifferential, ∂f, is just the set of all subgradients, ξ, at a particular point. This is a generalization of the gradient which collapses to the gradient, whenever f is differentiable. As a simple example, the subdifferential of $f(\beta) = |\beta|$, the absolute value function (which is non-differentiable at $\beta = 0$) is:

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

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 subdifferential. In particular we appeal to the following result from non-smooth analysis (Rockafellar, 1970):

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. Starting from some parameter vector, we compute the jth component of the subdifferential of the function (keeping all other components fixed):

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

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 $\boldsymbol{\Psi}$ (Recall that $\boldsymbol{\Psi}'$ is defined to be the matrix $\boldsymbol{\Psi}$ with diagonal entries set to zero). The second equation follows from the first as the subdifferential of a univariate 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 subdifferential of the non-differentiable absolute value function, and set $\Omega_j = 2(\boldsymbol{\Psi}'\boldsymbol{\beta})_j + \theta_j$ (and thus define the vector Ω to be the gradient of the purely differentiable part of the objective function), we obtain the subdifferential of the objective function, 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 $\boldsymbol{\Psi}$ 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 6.

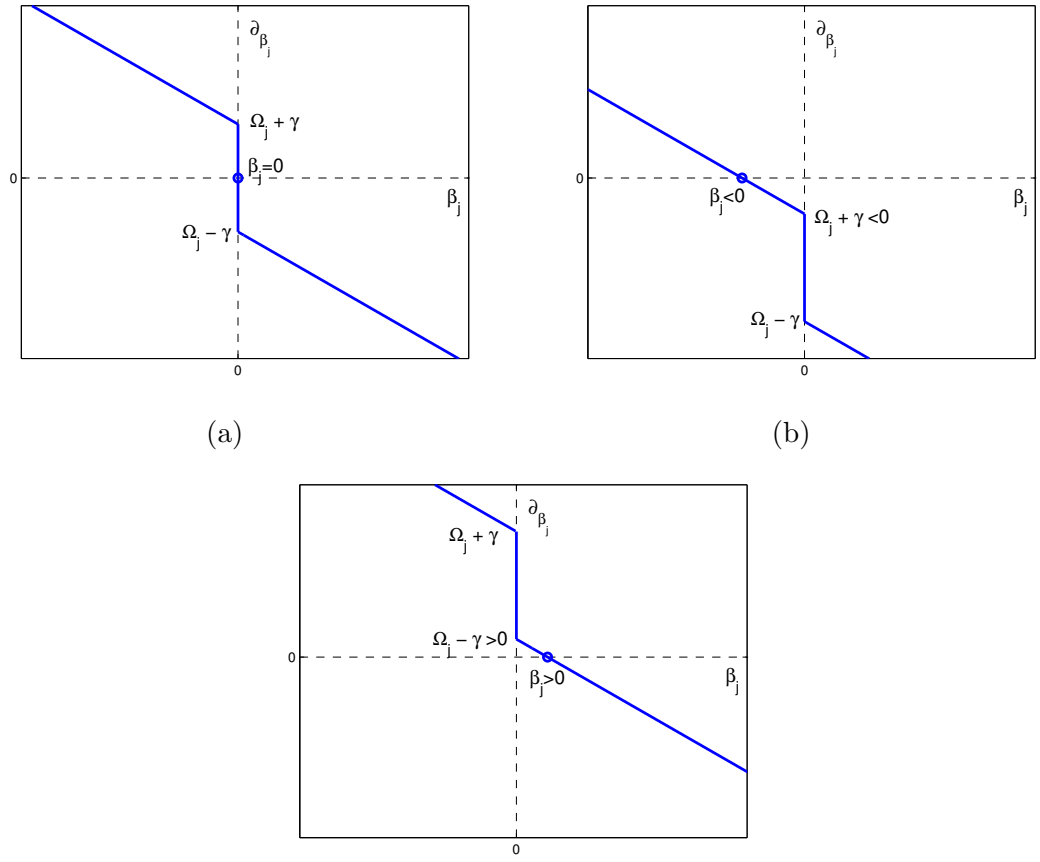


Figure 6: Illustration of cases occurring in the Shooting algorithm (a) If $|\Omega_j| \leq \gamma$ the constant portion of the subdifferential contains zero. In this case, set $\beta_j = 0$ (b) If instead, $\Omega_j < -\gamma$, the optimality conditions will be satisfied by setting $\beta_j = \frac{-\gamma - \Omega_j}{2\Psi_{jj}}$ (c) The case analogous to (b) but when $\Omega_j > \gamma$. Here the subdifferential is set equal to zero when $\beta_j = \frac{\gamma - \Omega_j}{2\Psi_{jj}}$.

Now to questions regarding the efficient implementation of the Shooting algorithm, used by the online, MP and RMMP algorithms. 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 $\mathbf{\Omega}$ updates efficiently for the RMMP algorithm, Algorithm 3. Recall that since we are discussing a multi-pass 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 this fact to show that in this case, you don't explicitly need the matrix $\mathbf{\Psi}$ (or $\tilde{\mathbf{\Psi}}$) to determine $\mathbf{\Omega}$. Indeed, after going through all the observations in the dataset (pass z, say):

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

which follows from the definitions of $\mathbf{\Omega}, \boldsymbol{\theta}$ and $\mathbf{\Psi}'$. In the above equation, $\mathrm{diag}(\mathbf{x}_i^2)$ 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 $O(kd)$.

Appendix C.

We present a proof sketch for the convergence behavior of the online algorithm in the infinite data limit. The intuition for is as follows: as $t \to \infty$, the Bayesian central limit theorems dictate that the posterior distribution tends (in distribution) to a multivariate Gaussian with ever shrinking covariance, (Bernardo and Smith, 1994). Thus, less and less information is required to encode the posterior distribution as more and more data is added—to a point. Indeed, in the limit, only the vector of the maximum likelihood value of the parameters, $\boldsymbol{\beta}_{MLE}$, is required to completely describe the posterior distribution.

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 gradient, 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 (Eskin 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.

E. Eskin, A. J. Smola, and S.V.N. Vishwanathan. Laplace propagation. In *Neural Information Processing Systems*, 16. MIT Press, 2003.

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, and D. Madigan. Large-scale bayesian logisitic regression for text categorization., 2007.

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.

K. Koh, S.-J. Kim, and S. Boyd. An interior-point method for large-scale l_1-regularized logistic regression. *Journal of Machine Learning Research*, 8:1519 – 1555, 2007.

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. In D. Saad, editor, *Online Learning in Neural Networks*, pages 363 – 378. Cambridge University Press, 1998.

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.

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

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.

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.

H. Zou. The adaptive lasso and its oracle properties. *J. Am. Stat. Assoc*, 101:1418 – 1429, 2006.