Bayesian Logistic Regression with Polya-Gamma latent variables

Kaspar Märtens Sherman Ip

October 21, 2015

Abstract

We have implemented an R package for Bayesian logistic regression, available in `https://github.com/kasparmartens/PolyaGamma`. To sample from the posterior distribution of the parameters, we use a data augmentation scheme which involves latent variables from the Polya-Gamma distribution.

1 Introduction

Logistic regression is a classical method for modeling data with a binary response. It can be used to detect associations between the binary outcome and the features of interest, as well as for predictive purposes. Examples of binary classification tasks include decisions whether an e-mail is spam or not, or whether a patient will get a heart disease, based on the available information such as cholesterol levels, and family history.

The classical approach for determining the parameters of the logistic regression model is to use maximum likelihood estimation. However, often it is favourable to work in a Bayesian framework, where we assign a prior distribution to the parameter vector. This can be beneficial even when there is no prior knowledge available, as the prior behaves as a regulariser and helps to avoid overfitting. Figure 1 illustrates how the posterior distribution of the parameter depends on the choice of prior.

However, due to the inconvenient form of likelihood, it is not straightforward to implement the Bayesian inference for this model. A data augmentation scheme for fitting a Bayesian probit regression was developed by Albert and Chib (1993), and since then there have been several attempts to use the same approach for logistic models, resulting in approximate inference. However, Polson et al. (2013) proposed an exact method for this, introducing latent variables from the Polya-Gamma family of distributions. The Polya-Gamma distributions were carefully constructed so that introducing latent variables from this family would yield a simple Gibbs sampler for the Bayesian logistic regression model.

This report will give a brief explanation on using latent Polya-Gamma variables to model Bayesian logistic regression followed by a description of our implementation. Experiments were conducted to investigate how our model performs on simulated data, real data and compared with the BayesLogit package.

2 Implementation

Let N be the number of data points, with binary outcomes $y_i \in \{0, 1\}$ and a set of covariates $x_i \in \mathbb{R}^p$. Suppose the data follows the logistic regression model

$$y_i \sim \text{Bernoulli}\left(\frac{1}{1+\exp(-x_i^T\boldsymbol{\beta})}\right)$$

and we have specified a prior distribution for the parameter vector with a prior distribution $\boldsymbol{\beta} \sim N(b, B)$.

In this report, we consider the prior distribution to have zero mean and a diagonal variance-covariance matrix, i.e. to be

$$\boldsymbol{\beta} \sim \mathrm{N}\left(0, \frac{1}{\lambda}\mathbf{I}\right) \tag{1}$$

where λ is the prior precision and can be treated as a tuning parameter. The smaller λ is, the less informative the prior is.

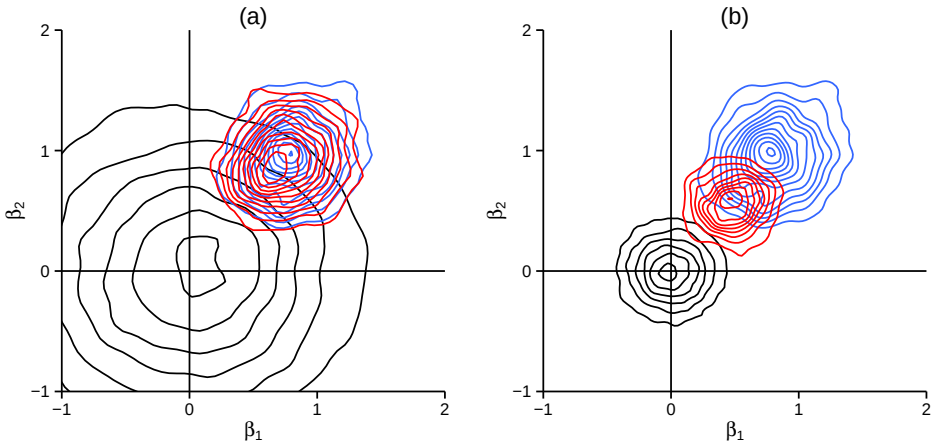


Figure 1: Illustration of the Bayesian logistic regression: the prior distribution (black), maximum likelihood estimator (blue), and posterior (red) shown by density plots in two different scenarios (a) and (b). Fixing $\beta_1 = 1$ and $\beta_2 = 1$, 100 data points were generated from the Bernoulli distribution with success probability $1/(1+\exp(-(\beta_1 x_1 + \beta_2 x_2)))$. The prior distribution is chosen as defined above with (a) $\lambda = 0.01$ and (b) $\lambda = 1$.

2.1 Data augmentation scheme and Gibbs sampling

Sampling from the posterior distribution of $\boldsymbol{\beta}$ can be achieved by introducing the auxiliary random variables $\omega_i, i = 1, ..., N$, and iterating the following two-step Gibbs sampling scheme:

1. $(\omega_i|\boldsymbol{\beta}) \sim \mathrm{PG}(1, x_i^T\boldsymbol{\beta})$
2. $(\boldsymbol{\beta}|y, \omega) \sim \mathrm{N}(m_\omega, V_\omega)$

where the first conditional distribution is a Polya-Gamma $PG(1, z)$ with some real number $z = x_i^T\boldsymbol{\beta}$ (see Figure 2 for examples), and the second one is a multivariate normal with the mean and covariance specified in Polson et al. (2013). Note that there is a latent variable ω_i for each data point, i.e. the first step needs to be implemented on each of the N data points, whereas the parameters $\boldsymbol{\beta}$ are sampled jointly as a vector.

One way for constructing a random variable X from a Polya-Gamma distribution $PG(1, z)$ with $z \in \mathbb{R}$ was according to the definition, i.e.

$$\sum_{k=1}^{\infty} \frac{g_k}{(k-0.5)^2 + \frac{z^2}{4\pi^2}} \tag{2}$$

where $g_k \sim \Gamma(1, 1)$ are i.i.d. random variables. The definition contains an infinite sum and it is not clear how its truncation to a finite number of terms will affect the results.

Instead, an accept-reject sampling procedure is proposed to sample from $PG(1, z)$.

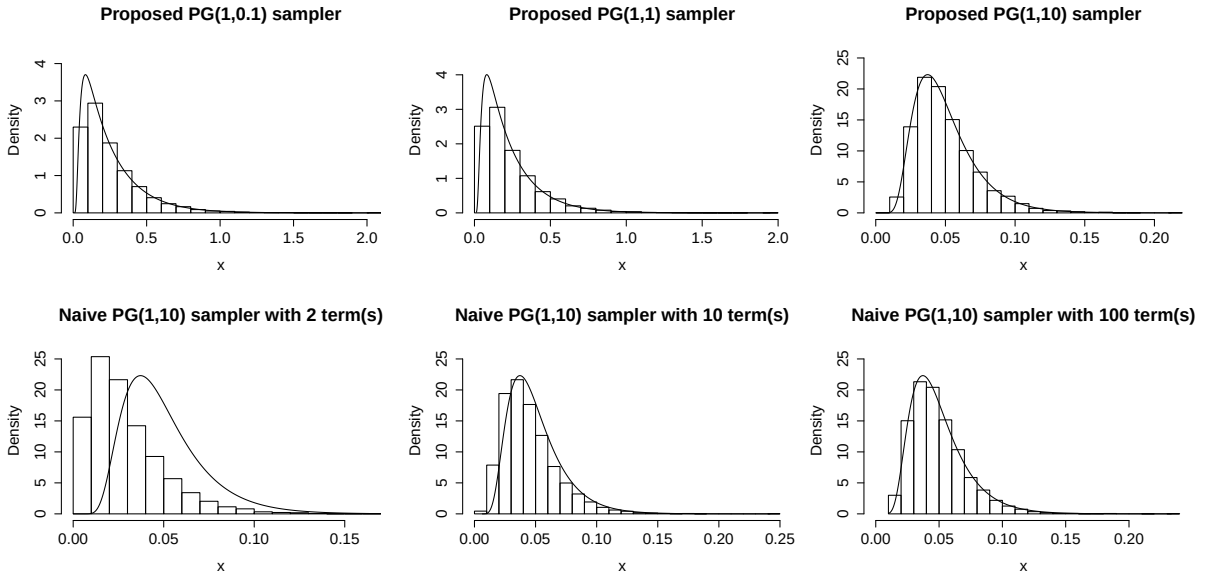


Figure 2: The 10,000 samples generated from the proposed (top row) and naive (bottom row) Polya-Gamma(1,z) sampler (histogram) plotted against the probability density function (curve).

2.2 Sampling from the Polya-Gamma distribution

Let $X \sim \mathrm{PG}(1, z)$, then the probability density function (p.d.f.) of X is given as

$$f_X(x) = \sqrt{\frac{2}{\pi x^3}} \cosh\left(\frac{z}{2}\right) \exp\left(-\frac{z^2 x}{2}\right) \sum_{n=0}^{\infty} (-1)^{-n} (n + 1/2) \exp\left[-\frac{(n+1/2)^2}{2x}\right] \quad \text{for } 0 < x. \tag{3}$$

Rejection sampling on this p.d.f. can be done by expressing it in a different form and in terms of the Jacobi distribution $\mathrm{J}^*(1, z)$. Suppose $J \sim \mathrm{J}^*(1, z/2)$ and has p.d.f. $f_J(.)$, then the two distributions are related by $X = \frac{J}{4}$. As a result, samples from $J \sim \mathrm{J}^*(z/2)$ can be used to generate samples of $X \sim \mathrm{PG}(1, z)$.

The p.d.f. of $J \sim \mathrm{J}^*(1, z)$ can be chosen to be expressed such that its partial sums have specific properties useful for rejection sampling. This can be done by setting the p.d.f. of J to be

$$f_J(x) = \cosh(z) \exp\left(-\frac{xz^2}{2}\right) \sum_{n=0}^{\infty} (-1)^n a_n(x) \quad \text{for } 0 < x \tag{4}$$

where the piecewise coefficients $a_n(x)$ are defined to be

$$a_n(x) = \begin{cases} \pi(n + 1/2) \left(\dfrac{2}{\pi x}\right)^{3/2} \exp\left[-\dfrac{2(n+1/2)^2}{x}\right] & \text{if } 0 < x \leq t \\ \pi(n + 1/2) \exp\left[-\dfrac{(n+1/2)^2 \pi^2 x}{2}\right] & \text{if } t < x \end{cases} \tag{5}$$

and $t = 0.64$ is the truncation point. Let $S_r(x)$ be the partial sums such that

$$S_r(x) = \cosh(z) \exp\left(-\frac{xz^2}{2}\right) \sum_{n=0}^{r} (-1)^n a_n(x) \ . \tag{6}$$

A useful property of the partial sums is that they fluctuate around the target density symmetrically such that

$$S_0(x) > S_2(x) > \cdots > f_J(x) > \cdots > S_3(x) > S_1(x) \ . \tag{7}$$

Because all $S_r(x)$ for even r are bigger than the target distribution $f_J(x)$, $S_0(x)$ can be used as the proposal distribution where

$$S_0(x) = \begin{cases} \frac{\pi\cosh(z)}{2}\left(\frac{2}{\pi x}\right)^{3/2}\exp\left(-\frac{xz^2}{2}-\frac{1}{2x}\right) & \text{if } 0 < x \leq t \\ \frac{\pi\cosh(z)}{2}\exp\left[-\left(\frac{z^2}{2}+\frac{\pi^2}{8}\right)x\right] & \text{if } t < x \end{cases} . \tag{8}$$

Figure 3 shows the density of the proposal (red and blue lines) and the target (black line) distribution together. They are indeed very close together and using such a proposal gives a very low rejection rate, no more than 9 rejections out of 10,000 samples, making it very efficient.

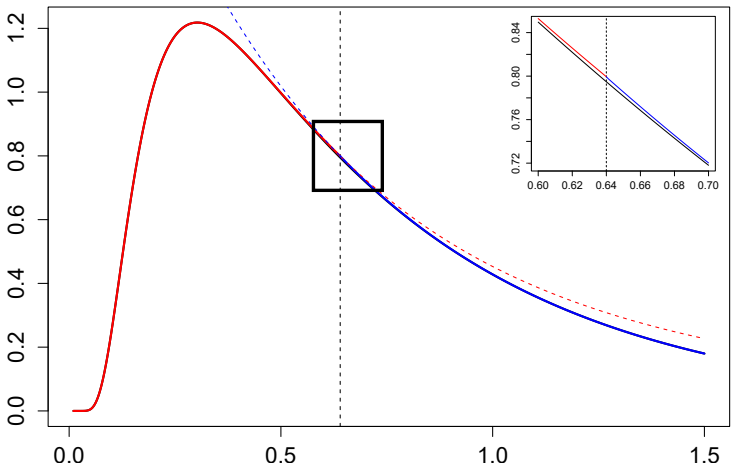


Figure 3: Visualisation of the accept-reject algorithm for the target PG(1, 1) distribution (density in black). The proposal distribution is defined in two pieces: for $x \in (0, 0.64]$ (density in red) and $x \in (0.64, \infty)$ (blue). The middle portion of the figure has been zoomed in (top right corner). The dashed lines (red and blue) extend the densities of proposal distributions outside their defined range.

Examining equation (8), we can see that the first line corresponds to the density of the truncated Inverse Normal distribution and the bottom line to the truncated exponential distribution. Thus, denoting

$$\Psi_1 \sim \text{InverseNormal}(|z|^{-1}, 1) \tag{9}$$
$$\Psi_2 \sim \text{Exp}(z^2/2 + \pi^2/8) \tag{10}$$

the p.d.f. of the proposal Y can be written as follows

$$f_Y(y) = \frac{1}{p+q}\left[f_{\Psi_1}(y)\mathbb{I}_{0<y<t} + f_{\Psi_2}(y)\mathbb{I}_{y>t}\right] \qquad \text{for } 0 < y\ , \tag{11}$$

where

$$p = \int_0^t f_{\Psi_1}(x)\mathrm{d}x \quad \text{and} \quad q = \int_t^\infty f_{\Psi_2}(x)\mathrm{d}x\ . \tag{12}$$

This results in a relatively simple algorithm for sampling a random variable Y from the proposal mixture.

- Generate y from the proposal distribution f_Y, i.e.
  - with probability $p/(p+q)$, sample y from the truncated InverseNormal$(|z|^{-1}, 1)$
  - with probability $q/(p+q)$ from the truncated Exp$(z^2/2 + \pi^2/8)$
- Apply the accept-reject approach. I.e. generate $u \sim \text{U}(0, y)$ and accept this if $u < f(y)$, otherwise reject. Note that we cannot compute $f(y)$ exactly, we are comparing u instead to the partial sums $S_r(y)$. If $u > S_r(y)$ for any even r, then $u > f_J(y)$ and the sample is rejected. Otherwise the sample is accepted if $u < S_r(y)$ for any odd r as this implies $u < f_J(y)$.

2.3 Example code

Our implementation is available in PolyaGamma package within the function gibbs_sampler. This can be used as follows

```
library(PolyaGamma)
data = generate_from_simple_logistic_model(n=100)
obj = gibbs_sampler(data$y, data$X, lambda=0.001, n_iter_total=150, burn_in=50)

## ================================================================================

obj

##
##   MCMC sample from the posterior distribution of beta.
##   Chain length: 100 (with 50 burn-in removed).
##   Number of parameters: 2.
##   Posterior means: 1.674, 1.586.
##   Posterior standard deviations: 0.348, 0.429.
##
```

3 Experiments and results

3.1 Tests on simulated data

As a proof of principle, we started by generating data from a simple logistic regression model, i.e. with a success probability $P(y_i = 1)$ modelled by $1/\exp(-(\beta_0 + \beta_1 x_i))$. We have fixed $\beta_0 = 1$ and $\beta_1 = 1$ to generate samples of varying size N. For all the following analysis we run the MCMC chain for 500 iterations and discarded the first 100 as burn-in.

Trace plots (Figure 4) indicate good mixing of the chains and the autocorrelations are relatively small. On the simple example the sampler seems to work efficiently. Next, we verified that for increasing sample size N, the posterior distribution of the parameter becomes more and more concentrated towards the true value (boxplots and the decreasing mean squared error in Figure 5 indicate this). Additionally, we verified the correctness of our implementation by estimating the posterior distributions with the BayesLogit package and comparing the results. As expected, these are highly similar, with small deviations likely arising from a different initialisation.

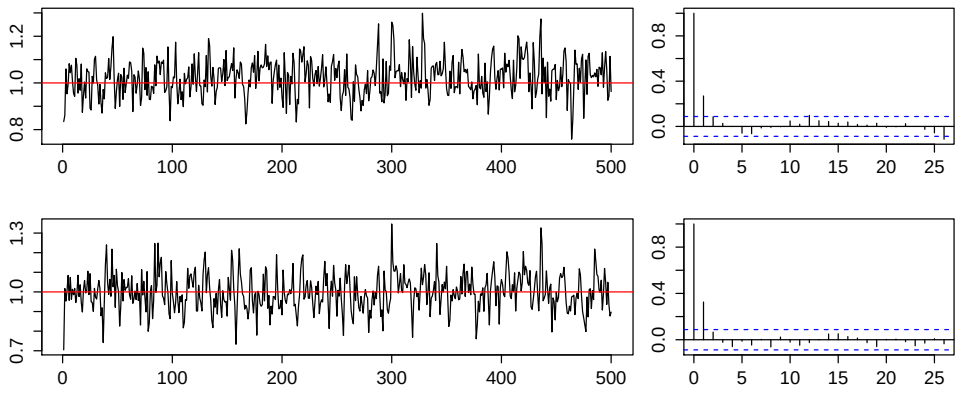


Figure 4: Traceplot (left) and autocorrelations (right) from the chains of β_0 (top row) and β_1 (bottom row) on simulated data. Horizontal red line denotes the true value.

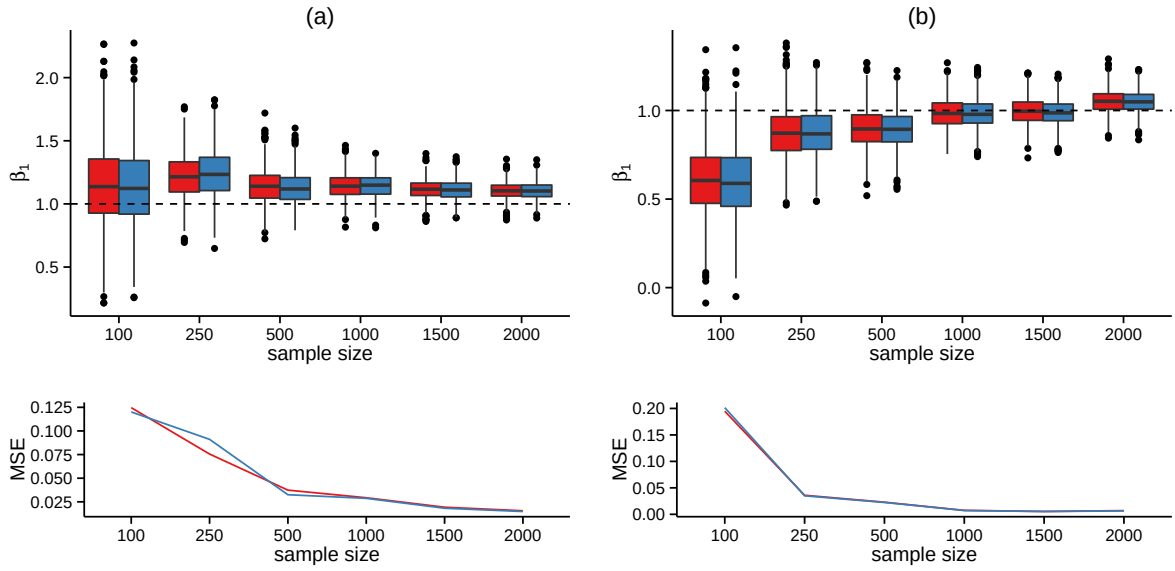


Figure 5: Samples from the posterior distribution of the parameter, comparing our implementation (red) and the BayesLogit package (blue), for a vague prior (a) $\lambda = 0.01$ and a strong one (b) $\lambda = 10$.

3.2 Tests on real data

Experiments were conducted to investigate how the Bayesian logistic regression can be used as a classifier on real datasets. The classifier was then compared, in prediction error, with the classical logistic regression.

Each dataset was randomly split into two equally sized disjoint subsets, the training and test set. The classifier was trained using the training set, and the training set only, so that it can be used to predict the responses given the feature vectors. The prior, used in the experiments, was set to be as non-informative as possible but can be varied univariately by changing λ. Because the precision matrix is a diagonal matrix with equal entries, the training sets were normalised to have zero mean and standard deviation of one, this was done so that the prior has an equal impact on all elements of the posterior $\boldsymbol{\beta}$. The predictive distribution, given a feature vector x, is the posterior expectation of the likelihood, that is

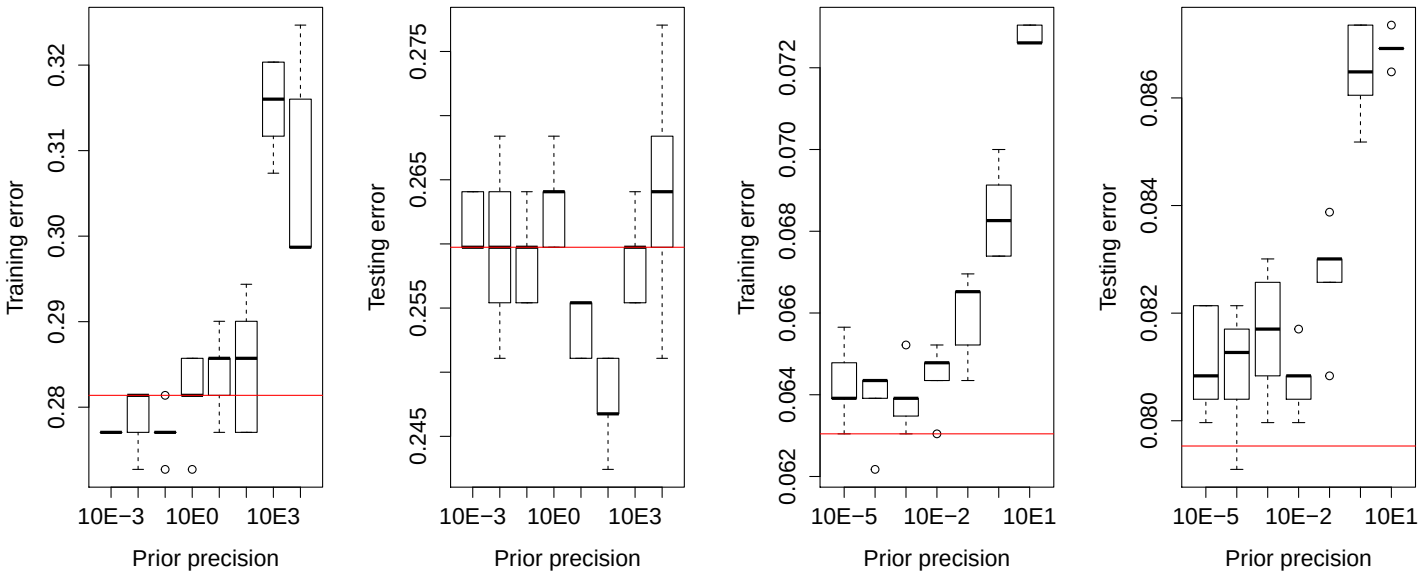
$$\mathrm{E}_{\boldsymbol{\beta}|\mathbf{Y}}\left[\frac{1}{1+\mathrm{e}^{-x^{\mathrm{T}}\boldsymbol{\beta}}}\right] . \tag{13}$$

Prediction of the response variable, given the feature vector x, was done by estimating the posterior expectation of the likelihood by taking a sample mean of the logistic function. In other words, after obtaining the samples $\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \cdots, \boldsymbol{\beta}_M$ from the posterior distribution, the prediction of the response $\widehat{y}$, given the feature vector x, is

$$\widehat{y} = \text{round}\left[\sum_{i=1}^{M}\frac{1}{1+\mathrm{e}^{-x^{\mathrm{T}}\boldsymbol{\beta}_i}}\right] . \tag{14}$$

As stated before, we used $M = 400$. This was repeated 5 times to get a sample of training and test errors.

Two datasets were investigated in the experiment. *SAheart* is a small dataset with sample size $N = 462$ and number of features (including an intercept) $p = 7$, while *spambase* is a large dataset with $N = 4601$ and $p = 58$. Figures 6a and 6b shows the training and test errors of the two datasets. With regards to *SAheart*, as expected the training error decreased for smaller, and less informative, prior precision. For $\lambda = 10^2$, the test error showed evidence that the Bayesian logistic regression outperforms the classical logistic regression. For the dataset *spambase*, the training error behaved as expected with smaller prior



(a) *SAheart* (b) *spambase*

Figure 6: Training and testing error on the dataset *SAheart* and *spambase* using the Bayesian logistic regression (box plot) and classical logistic regression (red line). The Bayesian logistic regression made predictions by using the sample mean of 400 posterior logistic regressions, this was repeated 5 times to get a sample of errors on the training and test set.

precision giving smaller training errors. However for this dataset, on average the test errors are larger compared to the results obtained from classical logistic regression.

The performance of the Bayesian logistic regression compared with the classical version were similar and may be better depending on the dataset. Nevertheless, Bayesian logistic regression may be a good candidate for a binary classifier, where selecting the value of λ which optimises the test error may be done by using cross validation methods.

3.3 Naive sampler for Polya-Gamma distribution

Instead of using the accept-reject method proposed in Polson et al. (2013), in principle one could simply truncate the infinite sum in (2). Thus we compared this naive approach with the more extensive simulation.

With a small (ten or smaller) number of terms, the truncated sum is not a good approximation of the Polya-Gamma distribution (Figure 2), but using 100 terms approximates the Polya-Gamma density quite well. Carrying out comparison of average computation time, we saw that compared to the proper accept-reject simulation, the approximation of 100 terms is four times faster.

We applied both the naive sampler and the proposed sampler to the real data sets, and compared the test error. For a fixed $\lambda = 0.001$, the results were comparable, whereas the naive approach provided slightly less accurate out-of-sample predictions. The test errors for *SAheart* data were 0.26 and 0.27 (exact vs. naive), and for *spambase* data 0.08 and 0.11 (exact vs naive). For the latter, we observed problems with the convergence of the chains, see an example in Figure 7.

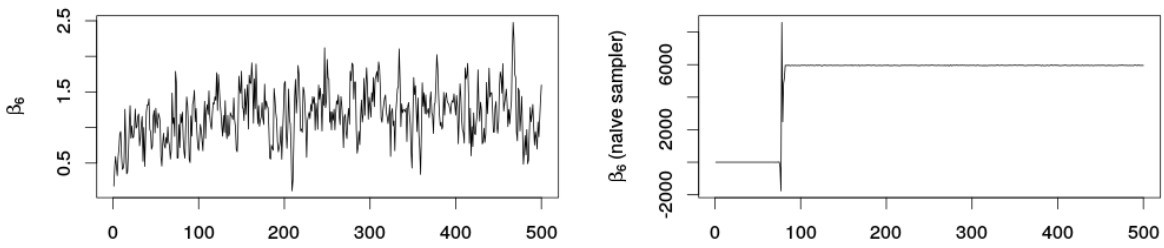


Figure 7: Naive Polya-Gamma sampler may produce a chain which converges to widely different values. Traceplot for the first 500 iterations for parameter β_6 from the spambase data, using either the proposed sampler (left) or the naive one (right).

3.4 Comparison with BayesLogit package

In section 3.1, we verified that our implementation provides highly similar posterior distribution as the BayesLogit package. Next, we compared the performance of our implementation with the one provided in BayesLogit package. Their approach is slightly different from ours, as they bin the data according to the features x_i, and aggregate the binary y_i values for each unique x_i. We note that this may reduce the number of latent variables ω_i.

The respective computation times over 5 repetitions are shown in Table 1. As BayesLogit package uses C calls, it is remarkably faster compared to our implementation (on the large *spambase* dataset, the average computation time 2 seconds versus 63 seconds). Using a naive Polya-Gamma sampler reduces computation time compared to our implementation of the proposed sampler, but as shown previously (Fig. 7), this approximation may be unreliable.

Table 1: Computation time in seconds for the two data sets studied. Chains were run for 500 iterations.

	SAheart			spambase		
	min	mean	max	min	mean	max
BayesLogit	0.06	0.06	0.07	1.64	1.69	1.71
Our implementation (naive)	1.50	1.55	1.65	19.42	19.61	19.78
Our implementation	5.50	5.56	5.60	62.95	63.25	63.70

References

James H Albert and Siddhartha Chib. Bayesian analysis of binary and polychotomous response data. *Journal of the American statistical Association*, 88(422):669–679, 1993.

Nicholas G Polson, James G Scott, and Jesse Windle. Bayesian inference for logistic models using pólya–gamma latent variables. *Journal of the American Statistical Association*, 108(504):1339–1349, 2013.