

# For Probabilistic Prediction, Full Bayes is Better than Point Estimators

Bob Carpenter

April 2019

## Abstract

A probabilistic prediction takes the form of a distribution over possible outcomes. With proper scoring rules such as log loss or square error, it is possible to evaluate such a probabilistic prediction against a true outcome. This short note provides simulation-based evaluation of full Bayesian inference, where we average over our estimation uncertainty, and two forms of point estimation, one that uses the posterior mode (max a posteriori) and one that uses the posterior mean (as is typical with variational inference). The example we consider is a simple Bayesian logistic regression with potentially correlated predictors and weakly informative priors. To make a long story short, full Bayes has lower expected log loss and squared error than either of the point estimators.

## Logistic Regression

### Observed data

The data consists of binary observations  $y_n \in \{0, 1\}$  paired with  $K$ -dimensional vectors of predictors<sup>1</sup>  $x_n$  for  $n \in 1 : N$ . That is,  $x \in \mathbb{R}^{N \times K}$  is the complete data matrix and  $x_n$  is its  $n$ -th row. We will assume the columns of  $x$  are standardized,<sup>2</sup> so that for every column  $k$  corresponding to a predictor, we have

$$\text{mean}(x_{1:N, k}) = 0$$

and

$$\text{sd}(x_{1:N, k}) = 1.$$

### Data distribution

For the purposes of simulation, we will assume the data have a multivariate normal distribution with a positive-definite covariance matrix  $\Sigma$ ,

$$x_n \sim \text{multinormal}(0, \Sigma).$$

We will assume the covariance matrix  $\Sigma$  has a unit diagonal so that each  $x_{n, k}$  has a standard normal marginal distribution.<sup>3</sup>

<sup>1</sup> Predictors are also called covariates or features.

<sup>2</sup> To standardize an unstandardized variable  $x = x_1, \dots, x_N$ , set

$$z(x) = \frac{x - \text{mean}(x)}{\text{sd}(x)}.$$

A standardized value  $z(x)$  of 1 corresponds to an original value  $x$  that is one standard deviation above the mean; a value of -2.5 corresponds to a value two and a half standard deviations below the mean.

<sup>3</sup> In symbols, if  $\mu = 0$  and  $\text{diag}(\Sigma) = 1$ , then  $y \sim \text{multinormal}(\mu, \Sigma)$  implies that marginally, each  $y_k \sim \text{normal}(0, 1)$ .

### Sampling distribution

The model is parameterized with an intercept  $\alpha \in \mathbb{R}$  and coefficient vector  $\beta \in \mathbb{R}^K$ . Logistic regression is a generalized linear model where the linear predictor

$$\alpha + x_n \beta = \alpha + \sum_{k=1}^K x_{n,k} \cdot \beta_k,$$

represents the log odds of  $y_n$  being equal to one.<sup>4</sup>

Given the log odds, the probability that  $y_n$  is one is given by inverting the log odds function,<sup>5</sup>

$$\Pr[y_n = 1] = \text{logit}^{-1}(\alpha + x_n \beta).$$

The sampling distribution is then defined to follow the log odds,

$$y_n \sim \text{bernoulli}(\text{logit}^{-1}(\alpha + x_n \beta)),$$

for observations indexed by  $n \in 1 : N$ . It is important to note that the  $y_n$  are defined by sampling according to the log odds.<sup>6</sup>

### Prior distribution

Given the logistic scale and standardized predictors, we assume weakly informative priors,<sup>7</sup>

$$\alpha, \beta_k \sim \text{normal}(0, 2),$$

for  $k \in 1 : K$ .

### Predictive distribution

Suppose we also have some observed test predictors  $\tilde{x}_n$  for  $n \in 1 : \tilde{N}$ , but we do not know the corresponding outcomes,  $\tilde{y}_n$ . These outcomes need to be predicted based on the predictors  $\tilde{x}_n$  and the knowledge of the values of the regression coefficients,  $\alpha$  and  $\beta$ . Although not strictly necessary, we will assume for the sake of simulation that  $\tilde{x}_n$  has the same distribution as the training predictors  $x$ .<sup>8</sup>

If we knew the parameter values  $(\alpha, \beta)$ , we would know that the distribution of  $\tilde{y}_n$  is just the sampling distribution,

$$\tilde{y}_n \sim \text{bernoulli}(\text{logit}^{-1}(\alpha + \tilde{x}_n \beta)).$$

### Inference

In general, we do not know the regression coefficients  $\alpha$  and  $\beta$ ; we only observe data  $(x, y)$  that lets us infer their values with some

<sup>4</sup> The function  $\text{logit} : (0, 1) \rightarrow (-\infty, \infty)$  maps a probability  $v \in [0, 1]$  to its log odds,

$$\text{logit}(v) = \log \frac{v}{1-v}.$$

<sup>5</sup> The inverse logit function  $\text{logit}^{-1} : (-\infty, \infty) \rightarrow (0, 1)$  maps a log odds value  $u \in (-\infty, \infty)$  to its corresponding probability,

$$\text{logit}^{-1}(u) = \frac{1}{1 + \exp(-u)}. \text{The inverse logit function is also known as the sigmoid function.}$$

<sup>6</sup> Because it is used as a decision process to choose a response, a common simulation mistake is to treat the log odds as a deterministic threshold and define

$$y_n = \begin{cases} 1 & \text{if } \text{logit}^{-1}(\alpha + x_n \beta) > \frac{1}{2} \\ 0 & \text{otherwise.} \end{cases}$$

Such an approach typically leads to separable data leading to infinite maximum likelihood coefficients.

As some indication of how common this mistake is in simulating logistic regression data, it comes up several times in the first page of hits for the Google query (simulate data with logistic regression), e.g., (1) (2), (3).

<sup>7</sup> Weakly informative priors provide the scale of a parameter; for logistic regression, log odds outside of the -5 to 5 range have very low or very high probabilities. For example,

$$\text{logit}^{-1}(6) \approx 0.9975.$$

So we keep the coefficients in the -3 to 3 range to match the predictors in the same range so that their products tend to not be too extreme on the log odds scale.

<sup>8</sup> If the conditions are not matched, poststratification can be used to make collective predictions, such as estimating accuracy, on mismatched data sets.

uncertainty. Similarly, we do not know the outcomes  $\tilde{y}_n$  we wish to predict based on their predictors  $\tilde{x}_n$  and the previously observed data  $(x, y)$ .

### Bayesian inference

Full Bayesian inference averages<sup>9</sup> over the uncertainty in our parameters given the data. Symbolically, this can be expressed in terms of a posterior distribution, as given by Bayes's rule,

$$\begin{aligned} p(\alpha, \beta \mid x, y) &= \frac{p(y \mid \alpha, \beta, x) \cdot p(\alpha, \beta)}{p(y)} \\ &\propto p(y \mid \alpha, \beta, x) \cdot p(\alpha, \beta). \end{aligned}$$

with terms for the sampling distribution  $p(y \mid \alpha, \beta, x)$  and prior  $p(\alpha, \beta)$ .

What we really want is inference for  $\tilde{y}$ , the unobserved outcomes. These are defined by taking a weighted average of predictions  $p(\tilde{y} \mid \tilde{x}, \alpha, \beta)$ , where the weights are determined by the posterior density  $p(\alpha, \beta \mid x, y)$ ,

$$p(\tilde{y} \mid \tilde{x}, x, y) = \int_{\mathbb{R}} \int_{\mathbb{R}^K} p(\tilde{y} \mid \tilde{x}, \alpha, \beta) \cdot p(\alpha, \beta \mid x, y) \, d\beta \, d\alpha.$$

Given posterior draws  $(\alpha^{(m)}, \beta^{(m)})$ ,<sup>10</sup> the posterior predictive distribution may be calculated by plugging in draws and averaging,

$$p(\tilde{y} \mid \tilde{x}, x, y) \approx \frac{1}{M} \sum_{m=1}^M p(\tilde{y} \mid \tilde{x}, \alpha^{(m)}, \beta^{(m)}).$$

Expected error in the estimate decreases as  $\mathcal{O}(1/\sqrt{M})$ .<sup>11</sup>

### Max a posterior approximate inference

The maximum a posteriori (MAP) estimator for parameters  $(\alpha, \beta)$  is given by

$$\alpha^*, \beta^* = \arg \max_{\alpha, \beta} p(\alpha, \beta \mid x, y).$$

It is common in machine learning applications to simply plug-in  $(\alpha^*, \beta^*)$  for inference, using the approximation

$$p(\tilde{y} \mid \tilde{x}, x, y) \approx p(\tilde{y} \mid \tilde{x}, \alpha^*, \beta^*).$$

The approximation arises because the values  $\alpha^*, \beta^*$  are being treated as certain when the data  $(x, y)$  provide only limited information about their values.

<sup>9</sup> For continuous quantities, averages of functions  $f(u)$  weighted by a probability function  $p(u)$  for a random variable  $U$  are given by expectations

$$\mathbb{E}[f(U)] = \int_U f(u) \cdot p(u) \, du.$$

<sup>10</sup> These may be (anti-)correlated, as in the draws produced by Markov chain Monte Carlo.

<sup>11</sup> Technically, the denominator the square root of the effective sample size, but this is linearly related to  $M$ , so the result holds as stated up to an order.

### Variational approximation inference

Variational Bayes (VB) attempts to find an approximate distribution matching the posterior. In machine learning settings, the goal is typically to extract point estimates corresponding to the approximate posterior mean values.<sup>12</sup> If the VB approximation were perfect, the optimizer will find the true posterior means.<sup>13</sup> For the purposes of simulation, we are going to suppose we have a good enough variational approximation to find the true posterior means.<sup>14</sup>

We can define the posterior means directly as conditional posterior expectations,

$$\begin{aligned}\hat{\alpha}, \hat{\beta} &= \mathbb{E}[\alpha, \beta \mid x, y] \\ &= \int_{\mathbb{R}} \int_{\mathbb{R}^K} (\alpha, \beta) \cdot p(\alpha, \beta \mid x, y) d\beta d\alpha.\end{aligned}$$

We can calculate the exact posterior means to arbitrary precision using simulated draws  $\alpha^{(m)}, \beta^{(m)}$  from the posterior,

$$\begin{aligned}\hat{\alpha} &= \frac{1}{M} \sum_{m=1}^M \alpha^{(m)} \\ \hat{\beta} &= \frac{1}{M} \sum_{m=1}^M \beta^{(m)}.\end{aligned}$$

Unlike maximum likelihood estimates, posterior mean estimates are unbiased and have the pleasant property of minimizing expected square error in the parameter estimates (we define square error below).

Given the posterior means, we can define another point-based approximation of the predictive distribution,

$$p(\tilde{y} \mid \tilde{x}, x, y) \approx p(\tilde{y} \mid \tilde{x}, \hat{\alpha}, \hat{\beta}).$$

As with posterior mode estimates, posterior mean estimates are approximate in that they assume that  $\hat{\alpha}, \hat{\beta}$  are estimated correctly, whereas there is residual uncertainty after observing the training data  $(x, y)$ .

### Simulation Experiment

In this section, we're going to simulate some correlated predictors and outcomes and then fit our logistic regression model with full Bayes, posterior modes, and posterior means for the purposes of evaluating predictive accuracy.

#### Stan program

Here's a Stan program implementing the logistic regression model.

<sup>12</sup> It's possible to extract posterior simulation draws given the full variational approximation, but that is almost never done other than in Stan. Typically, variational approximations are mean field (diagonal covariance), and so the posterior approximation is poor when parameters are correlated, as in typical logistic regression applications to language, vision, survey questions, or other correlated system of predictors. It is typical instead to just plug in the approximate posterior means found by variational inference.

<sup>13</sup> This doesn't matter if it uses a mean-field approximation or not; best case, it recovers the true posterior means.

<sup>14</sup> This is very unlikely in practice given the nature of the variational approximations, but given the variety of variational approximations possible and algorithms to fit them, this seemed like a good compromise for evaluation.

```

data {
  int<lower = 0> K;
  int<lower = 0> N;
  matrix[N, K] x;
  int y[N];

  int N_test;
  matrix[N_test, K] x_test;
  int y_test[N_test];
}
parameters {
  real alpha;
  vector[K] beta;
}
model {
  alpha ~ normal(0, 2);
  beta ~ normal(0, 2);
  y ~ bernoulli_logit(alpha + x * beta);
}
generated quantities {
  vector[N_test] E_y_test = inv_logit(alpha + x_test * beta);
  real log_loss = -bernoulli_logit_lpmf(y_test | alpha + x_test * beta);
  real sq_loss = dot_self(to_vector(y_test) - E_y_test);
}

```

The program includes not only the data and model declaration, but also the predictive distributions for square error and log loss. The true values of the test cases are provided to the program, but they are not used during training.<sup>15</sup>

### Simulated Data

We'll assume the predictor vectors are multivariate normal with  $K \times K$  covariance matrix  $\Sigma$  defined by

$$\Sigma_{i,j} = \rho^{|i-j|},$$

for some  $\rho \in (0, 1)$ .

Specifically, we'll take  $K = 50$ ,  $\rho = 0.9$ , and  $N = 'rN'$ . Here's what the matrix looks like for  $K = 5$  and  $\rho = 0.9$ .

$$\Sigma = \begin{bmatrix} 1 & 0.9 & 0.81 & 0.73 & 0.66 \\ 0.9 & 1 & 0.9 & 0.81 & 0.73 \\ 0.81 & 0.9 & 1 & 0.9 & 0.81 \\ 0.73 & 0.81 & 0.9 & 1 & 0.9 \\ 0.66 & 0.73 & 0.81 & 0.9 & 1 \end{bmatrix}$$

<sup>15</sup> Stan's generated quantities block is executed each iteration based on the parameter draws for that iteration; no information in the generated quantities block flows back to parameter estimation. For semi-supervised learning in cases such as naive Bayes or HMMs where there isn't the factorization of predictors found in logistic regression, we would need to move the tests into the model block to provide full Bayesian inference that also uses information in the test predictors (but not the test outcomes).

With highly correlated predictors, a predictor vector of size  $K$  provides less information than if the predictors were not correlated. How much less information depends on the degree of correlation—if two predictors are perfectly correlated or anti-correlated, there is no new information.

### *Loss Functions for Evaluating predictions*

Our goal is to provide predictive estimates of the probability that an unobserved outcome  $\tilde{y}_n$  takes value 1, given predictors  $\tilde{x}_n$  and training data  $(x, y)$ . In symbols, we want to estimate

$$\Pr [\tilde{y}_n = 1 \mid \tilde{x}_n, x, y].$$

To do so, we want to use a *proper scoring rule*.<sup>16</sup> We will consider three proper scoring functions (log loss, square loss, spherical loss) and one improper one (absolute loss).

<sup>16</sup> Proper scoring rules are ones which are optimized by the true distribution. Accuracy (aka 0/1 loss), as is commonly used in machine learning, is not a proper scoring rule.

#### *Log loss*

The simplest proper scoring rule is just the log probability (density or mass) of the true result under the model. For a given target  $y_n$  and probabilistic prediction  $\hat{y}_n$ , the log loss is defined by

$$\begin{aligned} \text{logloss}(y_n, \hat{y}_n) &= -\log \text{bernoulli}(y_n \mid \hat{y}_n) \\ &= (y_n = 1) ? -\log(\hat{y}_n) : -\log(1 - \hat{y}_n). \\ &= -\log(1 - |y_n - \hat{y}_n|). \end{aligned}$$

Extending to a complete test set  $y$  and response set  $\tilde{y}$ ,

$$\text{logloss}(y, \tilde{y}) = \sum_{n=1}^N \text{logloss}(y_n, \hat{y}_n).$$

We will report log loss as a rate per item after dividing by the number of test items,  $\tilde{N}$ .

If we think of our test set elements  $\tilde{y}_n$  as draws from the true distribution  $p(\tilde{y}_n \mid \tilde{x}_n)$ , then log loss provides a Monte Carlo estimate of the cross entropy rate from the true distribution of outcomes to the probabilistic forecasts.<sup>17</sup>

<sup>17</sup> Cross-entropy is defined from a density  $p$  to a density  $q$  as

$$\begin{aligned} H[p, q] &= -\int_Y p(y \mid x) \log q(y \mid x) dy, \\ &\approx -\frac{1}{\tilde{N}} \sum_{n=1}^{\tilde{N}} \log q(\tilde{y}_n \mid \tilde{x}_n). \end{aligned}$$

#### *Square error*

When the estimate takes a point form, as in our estimate of a probability, we can compute square loss,<sup>18</sup>

$$\text{sqloss}(y_n, \tilde{y}_n) = (y_n - \hat{y}_n)^2.$$

<sup>18</sup> Also known as a Brier score in machine learning after Glenn Brier, the statistician who introduced it in 1950.

Extending to an entire test set, the loss may be expressed compactly as

$$\begin{aligned} \text{sqloss}(y, \hat{y}) &= (y - \hat{y})^\top (y - \hat{y}) \\ &= \sum_{n=1}^{\tilde{N}} (y_n - \hat{y}_n)^2. \end{aligned}$$

Square loss is important in probability theory as it is the loss under which the average is the optimal estimator.<sup>19</sup> Put another way, the posterior mean is the point estimate that minimizes expected square loss.

Typically, when considering square error, we consider the square error rate by dividing by the number of test items, and then take the square root of that to put it back on the natural probability scale; the result is called “root mean square error” (RMSE),<sup>20</sup>

$$\text{rmse}(y, \hat{y}) = \sqrt{\frac{\text{sqloss}(y, \hat{y})}{\text{size}(y)}}$$

### Spherical loss

Given an outcome  $y_n$  and predicted probability  $\hat{y}_n$ , the spherical loss is

$$\text{sphere loss}(y_n, \hat{y}_n) = 1 - \frac{|y - \hat{y}|}{\sqrt{\hat{y}^2 + (1 - \hat{y})^2}}.$$

Extending to a whole data set, we just sum the spherical losses,

$$\text{sphere loss}(y, \hat{y}) = \sum_{n=1}^N \text{sphere loss}(y_n, \hat{y}_n).$$

Although this is a proper loss function, we will not be evaluating spherical loss because of the difficulty in interpretation—it is on the natural scale of  $y$ , but adjusted by the skew in the prediction. This reduces losses when predictive probabilities  $\hat{y}$  are near one half and increases losses when predictions are extreme (near zero or one).

### Absolute error

The final loss function we consider is improper, and corresponds to the absolute error, rather than square error. The absolute value function keeps the error positive,

$$\text{absloss}(y_n, \tilde{y}_n) = |\tilde{y}_n - \hat{y}_n|.$$

For multiple  $y_n$ , we sum the absolute losses. But we report them as an average given that the values are on the natural probability scale.

Absolute error is *not* a proper probabilistic scoring rule.

<sup>19</sup> In symbols, for  $x \in \mathbb{R}^N$ ,

$$\text{mean}(x) = \text{argmin}_u \sum_{n=1}^N (u - x_n)^2$$

<sup>20</sup> Taking means and square roots are monotonic operations and do not change the ordering of comparisons.

### Plotting the loss functions

The absolute error can range between 0 and 1. The square error has the same range. Log loss is minimized at zero, but unbounded as error approaches one. We can plot these functions

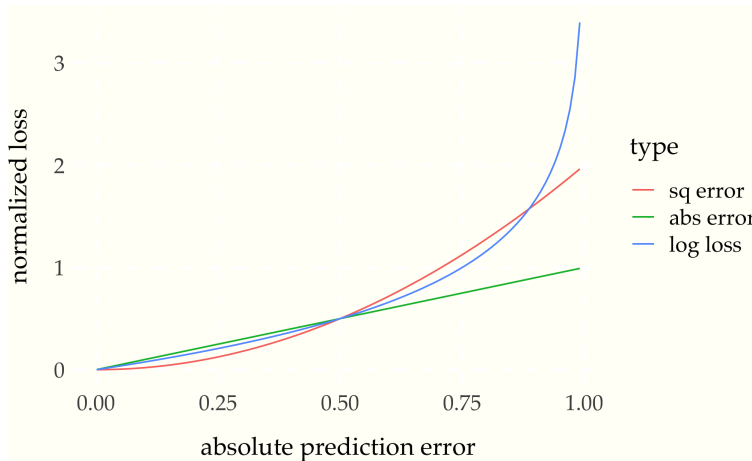


Figure 1: Three loss functions plotted as a function of absolute error. Because the dimensions of the loss functions vary and the scale is irrelevant for comparison, we have multiplied the loss functions so that they have loss one half for an absolute error of one half. For absolute errors between zero and one half, square error grows most slowly and absolute error increases the fastest. For errors above one half, square error grows fastest initially, then log loss takes over for very high absolute error (above ninety percent or so).

Each loss function has been multiplicatively normalized to provide a loss of one half for an absolute error of one half while leaving a loss of zero for an absolute error of zero.

It's also useful to observe the plot on the log scale to visualize the behavior of the loss functions at values near zero.

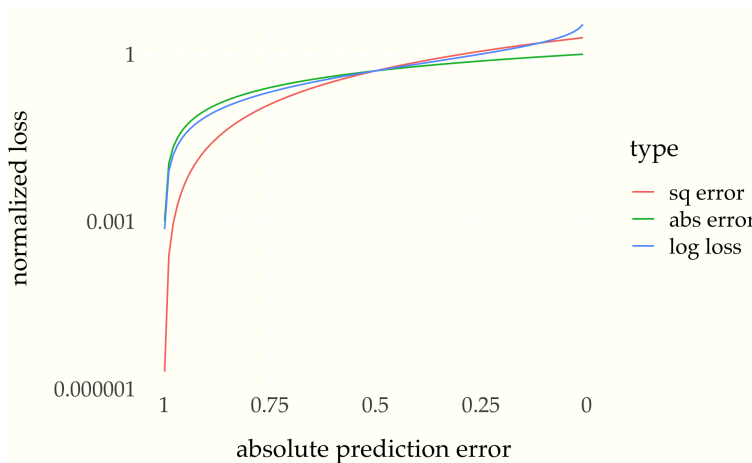


Figure 2: Log scale version of the previous plot of loss versus absolute error. The log scale highlights the differing behavior for small absolute errors.

### Evaluation with Stan

Fitting the model we provided earlier to our simulate data in Stan provides the following output, where we have only included the first coefficient,  $\beta_1$ .

Inference for Stan model: logistic-regression.  
 4 chains, each with iter=4000; warmup=2000; thin=1;  
 post-warmup draws per chain=2000, total post-warmup draws=8000.

	mean	se_mean	sd	1%	99%
alpha	-1.68	0.01	0.66	-3.3	-0.21
beta[1]	-0.12	0.01	1.14	-2.8	2.61
log_loss	294.53	0.47	38.44	216.2	392.90
sq_loss	53.83	0.05	4.77	43.6	65.60

	n_eff	Rhat
alpha	7997	1
beta[1]	8751	1
log_loss	6652	1
sq_loss	8607	1

Samples were drawn using NUTS(diag\_e) at Mon Apr 22 20:40:50 2019.  
 For each parameter, n\_eff is a crude measure of effective sample size,  
 and Rhat is the potential scale reduction factor on split chains (at  
 convergence, Rhat=1).

The 1% and 99% values provide the ends of the central 98% posterior interval on log loss and square loss for parameter values  $\alpha^{(m)}, \beta^{(m)}$ , drawn from the posterior. With only ' $rN$ ' training examples with correlated parameters, there is a great deal of uncertainty in the posteriors for coefficients and hence for predictions and for loss versus the test set. The range of log loss and square loss values shows the variability in those metrics across draws from the posterior.

### *Full Bayesian inference*

We first consider full Bayesian inference, where our estimate is derived from

$$p(\tilde{y}_n | \tilde{x}_n, x, y) = \int p(\tilde{y}_n | \tilde{x}_n, \alpha, \beta) \cdot p(\alpha, \beta | x, y) d(\alpha, \beta).$$

The result of running the simulation yields the following losses.

BAYES: sq error 40.20 root mean sq error 0.28  
 BAYES: log loss 133.26 log loss rate 0.27

The root mean square error being reported is the square root of the average square loss per test item. The log loss rate is just the log loss divided by the number of test items.

### Variational inference

Using the draws from the posterior, we calculate the posterior means  $(\hat{\alpha}, \hat{\beta})$  and then plug those in to use the estimate

$$p(\tilde{y}_n \mid \tilde{x}_n, \hat{\alpha}, \hat{\beta}).$$

The resulting errors are as follows.

```
VB: sq error 46.75  root mean sq error 0.31
VB: log loss 224.08  log loss rate 0.45
```

### Max a posteriori inference

Here, we use Stan's optimizing function to find the maximum a posteriori (MAP) estimates  $\alpha^*, \beta^*$  and proceed to plug those in for inference,

$$p(\tilde{y}_n \mid \tilde{x}_n, \alpha^*, \beta^*).$$

The resulting errors are as follows.

```
MAP: sq error 44.34  root mean sq error 0.30
MAP: log loss 165.34  log loss rate 0.33
```

### Probabilistic Training and Test Data

Traditionally, we estimate (i.e., train) logistic regression models based on dichotomous observations

$$y_n \in \{0, 1\}.$$

Now suppose instead that the outcomes  $y_n$  are not known exactly, but only with some uncertainty. That is, rather than knowing  $y_n$ , we only have a probability estimate of whether  $y_n = 1$  or  $y_n = 0$ . Such uncertainty can arise from noisy measurements of the true  $y_n$  value. For example, the data may be crowdsourced human data coding,<sup>21</sup> or it may arise from the application of imprecise heuristics. Alternatively, we may have uncertainty propagated from measurement devices such as RNA-seq readers of genomic sequences.

The math all generalizes if we assume that instead of dichotomous outcomes  $y_n \in \{0, 1\}$ , we have probabilistic outcomes,

$$\phi_n = \Pr[y_n = 1 \mid x_n].$$

Typically these  $\phi_n$  are the estimates from another probabilistic model, such as a noisy measurement model of human data coding or a model of fluorescence response and image processing steps for RNA-seq data.<sup>22</sup>

<sup>21</sup> Also known as data rating, annotation, and labeling.

<sup>22</sup> Ideally, the predictive variables  $\phi$  estimated by the training data measurement error model and the parameters  $\alpha, \beta$  of a regression based on that model will be modeled and estimated jointly.

*Weighted regression*

Training is simple with weighted observations. The log likelihood function for dichotomous data  $y_n \in \{0, 1\}$  is as defined previously,

$$\begin{aligned}\log p(y \mid x, \alpha, \beta) &= \log \prod_{n=1}^N p(y_n \mid x_n, \alpha, \beta) \\ &= \sum_{n=1}^N \log \text{bernoulli}(y_n \mid \text{logit}^{-1}(\alpha + x_n \beta)).\end{aligned}$$

To generalize to continuous outcomes  $\phi_n$ , we work in expectation with respect to the  $\phi_n$ . So instead of the likelihood, we use a weighted version,

$$\sum_{n=1}^N \phi_n \cdot \log \text{bernoulli}(1 \mid \text{logit}^{-1}(\alpha + x_n \beta)) + (1 - \phi_n) \cdot \log \text{bernoulli}(0 \mid \text{logit}^{-1}(\alpha + x_n \beta)).$$

This quantity simply replaces the likelihood in the model. If the values of  $\phi_n$  are 0 or 1, it reduces to our original likelihood function.

In Stan, our original vectorized sampling statement,

```
y ~ bernoulli_logit(alpha + x * beta);
```

is equivalent to the unrolled version

```
for (n in 1:N)
  target += bernoulli_logit(y[n] | alpha + x[n] * beta);
```

The weighted version computes the density as the expected log likelihood given probability estimates  $\phi_n = \Pr[y_n = 1 \mid x_n]$ ,

```
for (n in 1:N) {
  target += phi[n] * bernoulli_logit(1 | alpha + x[n] * beta);
  target += (1 - phi[n]) * bernoulli_logit(0 | alpha + x[n] * beta);
}
```

It's important that both target increment statements are used. The effect of incrementing both 0 and 1 results in proportion to their weights provides a form of data-driven regularization, where the resulting density is maximized when

$$\phi_n = \text{logit}^{-1}(\alpha + x_n \beta).$$

That is, we penalize coefficients for predicting probabilities for  $y_n$  that are too far away from the training set probabilities  $\phi_n$ . If we had sampled rather than weighted, the result would be the same in the limit of increasing amounts of data.

The resulting formulation is not properly generative—there is no way to generate the “observations”  $\phi_n$  from the regression coefficients  $\alpha, \beta$  and the observed predictors  $x_n$ .

In this particular case, because we know that the log odds are given by  $\text{logit}(\phi_n)$ , instead of using a weighted logistic regression, we can simply train a standard linear regression

$$\text{logit}(\phi_n) \sim \text{normal}(\alpha + x_n \beta, \sigma)$$

and use the coefficients  $\alpha, \beta$  predictively for new data with a logit link function,

$$\tilde{y}_n \sim \text{bernoulli}(\text{logit}^{-1}(\alpha + \tilde{x}_n \beta)).$$

### *Weighted logistic regression with Stan*

Here's the full Stan model for weighted logistic regression. It follows the description in the previous section. The data has been modified to read in a vector of probabilities `phi[1:N]` rather than a binary array `y[1:N]`.

```
data {
  int<lower = 0> K;
  int<lower = 0> N;
  matrix[N, K] x;
  vector[N] phi;
}
transformed data {
  vector[N] inv_logit_phi = inv_logit(phi);
}
parameters {
  real alpha;
  vector[K] beta;
}
model {
  vector[N] log_odds = alpha + x * beta;
  alpha ~ normal(0, 2);
  beta ~ normal(0, 2);
  for (n in 1:N) {
    target += inv_logit_phi[n] * bernoulli_logit_lpmf(1 | log_odds[n]);
    target += (1 - inv_logit_phi[n]) * bernoulli_logit_lpmf(0 | log_odds[n]);
  }
}
```

What we want to compare is how well we can estimate the model with draws from the distribution given by  $\phi$  versus estimating it using the  $\phi$  directly. We will again simulate  $N = 200$  data points, but this time rather than drawing  $y_n \sim \text{bernoulli}(\text{logit}^{-1}(\alpha + x_n \beta))$ , we will add noise to our estimates of  $\phi_n$ ,<sup>23</sup>

$$\phi_n = \alpha + x_n \beta + \epsilon_n$$

<sup>23</sup> If we do not add noise, the values of  $\alpha$  and  $\beta$  will be exactly identified if  $N > K + 1$  and the predictors are not perfectly correlated.

where

$$\epsilon_n \sim \text{normal}(0, 0.25).$$

which is roughly give or take 10% for a value of 50% when converted back to the probability scale; a noise scale of 0.5 rather than 0.25 would lead to an error of plus or minus 20% or so.

Here are the results of estimating with these approaches compared to the true values.

```

true: alpha =  0.14  beta[1] =  0.12  beta[2] =  0.32
weighted: alpha =  0.15  beta[1] =  0.22  beta[2] =  0.28
sampled: alpha = -0.30  beta[1] =  0.99  beta[2] =  2.46

```

Even with that much noise added to the log odds in the weighted case, the estimates from a training set of size 200 are much better, as can be seen from the handful of coefficients reported above.

The moral of this story is that binary data is very weak, but probabilistic data is much stronger.

### *Weighted evaluation with scoring functions*

All of our loss functions generalize to compute expected loss when faced with data of the form  $\phi_n = \Pr[y_n = 1 \mid x_n, \alpha, \beta]$ . For example, given the linear estimate  $\hat{y}_n = \alpha + x_n \beta$  of the log odds that  $y_n = 1$ , all of our loss functions are minimized when  $\phi_n = \text{logit}^{-1}(\alpha + x_n \beta)$ .

This transition from evaluation on a single outcome to weighted evaluations where the truth has a probability is commonly used in evaluating forecasts. The idea is that there's a true uncertainty of  $y_n$  given the predictors  $x_n$ , and we want to evaluate whether our inference system is estimating this uncertain properly. Lifting the scoring rules to deal with probabilistic test data yields general scoring functions as are used in the probabilistic forecasting literature. In the simplest case of binary distributions, the general scoring function corresponds to estimating the cross-entropy rate from the true probability distribution  $\text{bernoulli}(\phi_n)$  to the estimated one  $\text{bernoulli}(\hat{y}_n)$ . That is, the log loss rate estimates the cross-entropy rate, which is the cost of coding variables  $y_n$  using estimates  $\hat{y}_n$  which given that log loss is a proper scoring rule, is minimized when  $\hat{y}_n = \phi_n$ .

### *Conclusion*

When the model is well specified for the problem and there is not a huge amount of data, as in our logistic regression example, it's much more accurate to use Bayesian inference. With large amounts of data relative to the number of predictors, approximate methods perform almost as well as full Bayes.

One may argue that “big data” abound in real applications and “little data” should not be a problem. In reality, much of the big data we have consists of lots of little data. We may read millions of base pairs, but the amount of data we might have about two splice variants of interest is typically on the scale of the simulations done in this paper. Similarly, if we look at e-commerce, we have tons of data, but relatively little data about individual products or customers, as new users and items are continually added and old behaviors change.

To summarize the quantitative results in this short note, the error rates for a training set of size 200 with 50 correlated predictors are summarized in the following table.

	Inference	root mean square error	log loss rate
	Full Bayes	0.28	0.27
MAP (posterior mode)		0.3	0.33
VB (posterior mean)		0.31	0.45

The result from taking a single posterior draw  $(\alpha^{(m)}, \beta^{(m)})$  as a point estimate depends heavily on the draw, but almost all such draws are worse than the posterior mode and the posterior mean.

### *Appendix A: Some alternative evals*

We also include a table of evaluation results for full Bayes versus plugging in either the posterior mode (MAP) or mean (VB). We only run a single simulation for each set of conditions as it provides an overall feel for the trend of the evaluations.<sup>24</sup>

	estimator	N	K	rho	log_loss_rate	rmse
1	Bayes	8	2	0.0	0.89	0.57
2	MAP	8	2	0.0	0.92	0.57
3	VB	8	2	0.0	1.06	0.59
4	Bayes	8	2	0.9	0.82	0.51
5	MAP	8	2	0.9	0.90	0.51
6	VB	8	2	0.9	1.06	0.52
7	Bayes	8	8	0.0	0.66	0.48
8	MAP	8	8	0.0	0.88	0.54
9	VB	8	8	0.0	0.92	0.53
10	Bayes	8	8	0.9	0.99	0.61
11	MAP	8	8	0.9	1.37	0.66
12	VB	8	8	0.9	1.57	0.67
13	Bayes	32	2	0.0	0.71	0.51
14	MAP	32	2	0.0	0.71	0.51

<sup>24</sup> Combining simulations is tricky as each simulated condition has a different underlying entropy and hence different lower bound on cross-entropy and hence log loss.

estimator	N	K	rho	log_loss_rate	rmse	mae
Bayes	8	2	0.0	0.89	0.57	0.51
MAP	8	2	0.0	0.92	0.57	0.50
VB	8	2	0.0	1.06	0.59	0.51
Bayes	8	2	0.9	0.82	0.51	0.39
MAP	8	2	0.9	0.90	0.51	0.39
VB	8	2	0.9	1.06	0.52	0.37
Bayes	8	8	0.0	0.66	0.48	0.43
MAP	8	8	0.0	0.88	0.54	0.44
VB	8	8	0.0	0.92	0.53	0.41
Bayes	8	8	0.9	0.99	0.61	0.57
MAP	8	8	0.9	1.37	0.66	0.58
VB	8	8	0.9	1.57	0.67	0.59
Bayes	32	2	0.0	0.71	0.51	0.46
MAP	32	2	0.0	0.71	0.51	0.46
VB	32	2	0.0	0.72	0.51	0.46
Bayes	32	2	0.9	0.63	0.47	0.43
MAP	32	2	0.9	0.63	0.47	0.43
VB	32	2	0.9	0.63	0.47	0.42
Bayes	32	8	0.0	1.02	0.60	0.54
MAP	32	8	0.0	1.09	0.61	0.54
VB	32	8	0.0	1.27	0.64	0.55
Bayes	32	8	0.9	0.72	0.51	0.49
MAP	32	8	0.9	0.73	0.51	0.49
VB	32	8	0.9	0.75	0.52	0.48
Bayes	32	32	0.0	0.65	0.47	0.40
MAP	32	32	0.0	1.01	0.52	0.36
VB	32	32	0.0	1.76	0.55	0.35
Bayes	32	32	0.9	0.59	0.45	0.35
MAP	32	32	0.9	0.76	0.49	0.34
VB	32	32	0.9	0.98	0.50	0.31
Bayes	128	2	0.0	0.60	0.45	0.42
MAP	128	2	0.0	0.60	0.45	0.42
VB	128	2	0.0	0.60	0.45	0.42
Bayes	128	2	0.9	0.54	0.43	0.37
MAP	128	2	0.9	0.54	0.43	0.37
VB	128	2	0.9	0.54	0.43	0.37
Bayes	128	8	0.0	0.44	0.38	0.29
MAP	128	8	0.0	0.44	0.38	0.29
VB	128	8	0.0	0.44	0.38	0.28
Bayes	128	8	0.9	0.67	0.47	0.38
MAP	128	8	0.9	0.68	0.47	0.38
VB	128	8	0.9	0.69	0.47	0.37
Bayes	128	32	0.0	0.88	0.50	0.32
MAP	128	32	0.0	1.10	0.51	0.32
VB	128	32	0.0	1.49	0.52	0.31
Bayes	128	32	0.9	0.50	0.37	0.24
MAP	128	32	0.9	0.54	0.38	0.23
VB	128	32	0.9	0.61	0.38	0.21

Table 2: Log loss rate, root mean square error (rmse), and mean absolute error (mae) for full Bayes and plug-in estimates with posterior mode (MAP) and mean (VB) given  $N$  observations,  $K$  predictors, and a base  $\rho$  correlation between adjacent predictors.

Table 3: Foo

estimator	N	K	rho	log_loss_rate	rmse	mae
Bayes	8	2	0.0	0.89	0.57	0.51
MAP	8	2	0.0	0.92	0.57	0.50
VB	8	2	0.0	1.06	0.59	0.51
Bayes	8	2	0.9	0.82	0.51	0.39
MAP	8	2	0.9	0.90	0.51	0.39
VB	8	2	0.9	1.06	0.52	0.37
Bayes	8	8	0.0	0.66	0.48	0.43
MAP	8	8	0.0	0.88	0.54	0.44
VB	8	8	0.0	0.92	0.53	0.41
Bayes	8	8	0.9	0.99	0.61	0.57
MAP	8	8	0.9	1.37	0.66	0.58
VB	8	8	0.9	1.57	0.67	0.59
Bayes	32	2	0.0	0.71	0.51	0.46
MAP	32	2	0.0	0.71	0.51	0.46
VB	32	2	0.0	0.72	0.51	0.46
Bayes	32	2	0.9	0.63	0.47	0.43
MAP	32	2	0.9	0.63	0.47	0.43
VB	32	2	0.9	0.63	0.47	0.42
Bayes	32	8	0.0	1.02	0.60	0.54
MAP	32	8	0.0	1.09	0.61	0.54
VB	32	8	0.0	1.27	0.64	0.55
Bayes	32	8	0.9	0.72	0.51	0.49
MAP	32	8	0.9	0.73	0.51	0.49
VB	32	8	0.9	0.75	0.52	0.48
Bayes	32	32	0.0	0.65	0.47	0.40
MAP	32	32	0.0	1.01	0.52	0.36
VB	32	32	0.0	1.76	0.55	0.35
Bayes	32	32	0.9	0.59	0.45	0.35
MAP	32	32	0.9	0.76	0.49	0.34
VB	32	32	0.9	0.98	0.50	0.31
Bayes	128	2	0.0	0.60	0.45	0.42
MAP	128	2	0.0	0.60	0.45	0.42
VB	128	2	0.0	0.60	0.45	0.42
Bayes	128	2	0.9	0.54	0.43	0.37
MAP	128	2	0.9	0.54	0.43	0.37
VB	128	2	0.9	0.54	0.43	0.37
Bayes	128	8	0.0	0.44	0.38	0.29
MAP	128	8	0.0	0.44	0.38	0.29
VB	128	8	0.0	0.44	0.38	0.28
Bayes	128	8	0.9	0.67	0.47	0.38
MAP	128	8	0.9	0.68	0.47	0.38
VB	128	8	0.9	0.69	0.47	0.37
Bayes	128	32	0.0	0.88	0.50	0.32
MAP	128	32	0.0	1.10	0.51	0.32
VB	128	32	0.0	1.49	0.52	0.31
Bayes	128	32	0.9	0.50	0.37	0.24
MAP	128	32	0.9	0.54	0.38	0.23
VB	128	32	0.9	0.61	0.38	0.21

15	VB	32	2	0.0	0.72	0.51
16	Bayes	32	2	0.9	0.63	0.47
17	MAP	32	2	0.9	0.63	0.47
18	VB	32	2	0.9	0.63	0.47
19	Bayes	32	8	0.0	1.02	0.60
20	MAP	32	8	0.0	1.09	0.61
21	VB	32	8	0.0	1.27	0.64
22	Bayes	32	8	0.9	0.72	0.51
23	MAP	32	8	0.9	0.73	0.51
24	VB	32	8	0.9	0.75	0.52
25	Bayes	32	32	0.0	0.65	0.47
26	MAP	32	32	0.0	1.01	0.52
27	VB	32	32	0.0	1.76	0.55
28	Bayes	32	32	0.9	0.59	0.45
29	MAP	32	32	0.9	0.76	0.49
30	VB	32	32	0.9	0.98	0.50
31	Bayes	128	2	0.0	0.60	0.45
32	MAP	128	2	0.0	0.60	0.45
33	VB	128	2	0.0	0.60	0.45
34	Bayes	128	2	0.9	0.54	0.43
35	MAP	128	2	0.9	0.54	0.43
36	VB	128	2	0.9	0.54	0.43
37	Bayes	128	8	0.0	0.44	0.38
38	MAP	128	8	0.0	0.44	0.38
39	VB	128	8	0.0	0.44	0.38
40	Bayes	128	8	0.9	0.67	0.47
41	MAP	128	8	0.9	0.68	0.47
42	VB	128	8	0.9	0.69	0.47
43	Bayes	128	32	0.0	0.88	0.50
44	MAP	128	32	0.0	1.10	0.51
45	VB	128	32	0.0	1.49	0.52
46	Bayes	128	32	0.9	0.50	0.37
47	MAP	128	32	0.9	0.54	0.38
48	VB	128	32	0.9	0.61	0.38
49	Bayes	128	128	0.0	0.53	0.42
50	MAP	128	128	0.0	0.76	0.44
51	VB	128	128	0.0	2.20	0.47
52	Bayes	128	128	0.9	0.35	0.33
53	MAP	128	128	0.9	0.44	0.33
54	VB	128	128	0.9	0.90	0.35
55	Bayes	512	2	0.0	0.64	0.47
56	MAP	512	2	0.0	0.64	0.47
57	VB	512	2	0.0	0.65	0.47
58	Bayes	512	2	0.9	0.70	0.50

59	MAP	512	2	0.9	0.70	0.50
60	VB	512	2	0.9	0.70	0.50
61	Bayes	512	8	0.0	0.50	0.41
62	MAP	512	8	0.0	0.50	0.41
63	VB	512	8	0.0	0.50	0.41
64	Bayes	512	8	0.9	0.47	0.39
65	MAP	512	8	0.9	0.47	0.39
66	VB	512	8	0.9	0.47	0.39
67	Bayes	512	32	0.0	0.42	0.37
68	MAP	512	32	0.0	0.42	0.37
69	VB	512	32	0.0	0.44	0.37
70	Bayes	512	32	0.9	0.37	0.35
71	MAP	512	32	0.9	0.37	0.35
72	VB	512	32	0.9	0.38	0.36
73	Bayes	512	128	0.0	0.41	0.36
74	MAP	512	128	0.0	0.61	0.39
75	VB	512	128	0.0	1.02	0.39
76	Bayes	512	128	0.9	0.67	0.42
77	MAP	512	128	0.9	0.90	0.45
78	VB	512	128	0.9	1.22	0.45

mae

1	0.51
2	0.50
3	0.51
4	0.39
5	0.39
6	0.37
7	0.43
8	0.44
9	0.41
10	0.57
11	0.58
12	0.59
13	0.46
14	0.46
15	0.46
16	0.43
17	0.43
18	0.42
19	0.54
20	0.54
21	0.55
22	0.49
23	0.49

24 0.48  
25 0.40  
26 0.36  
27 0.35  
28 0.35  
29 0.34  
30 0.31  
31 0.42  
32 0.42  
33 0.42  
34 0.37  
35 0.37  
36 0.37  
37 0.29  
38 0.29  
39 0.28  
40 0.38  
41 0.38  
42 0.37  
43 0.32  
44 0.32  
45 0.31  
46 0.24  
47 0.23  
48 0.21  
49 0.35  
50 0.26  
51 0.24  
52 0.22  
53 0.17  
54 0.15  
55 0.42  
56 0.42  
57 0.42  
58 0.49  
59 0.49  
60 0.49  
61 0.33  
62 0.33  
63 0.33  
64 0.32  
65 0.32  
66 0.32  
67 0.23

```

68 0.23
69 0.23
70 0.24
71 0.24
72 0.23
73 0.20
74 0.19
75 0.18
76 0.24
77 0.25
78 0.24

```

Do I need something afterward here?

## *Appendix B: R Session Information*

```

R version 3.5.0 (2018-04-23)
Platform: x86_64-apple-darwin15.6.0 (64-bit)
Running under: macOS High Sierra 10.13.6

Matrix products: default
BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats      graphics  grDevices  utils
[5] datasets  methods   base

other attached packages:
[1] tufte_0.4          rstan_2.18.1
[3] StanHeaders_2.18.0 MASS_7.3-49
[5] magrittr_1.5       kableExtra_1.1.0
[7] knitr_1.20         ggplot2_3.1.0
[9] rmarkdown_1.10

loaded via a namespace (and not attached):
[1] tinytex_0.5        xfun_0.1
[3] tidyselect_0.2.4   purrr_0.2.5
[5] colorspace_1.3-2   htmltools_0.3.6
[7] stats4_3.5.0       viridisLite_0.3.0
[9] loo_2.0.0          yamll_2.2.0

```

[11] base64enc_0.1-3	rlang_0.2.1
[13] pkgbuild_1.0.2	pillar_1.2.3
[15] glue_1.2.0	withr_2.1.2
[17] bindrcpp_0.2.2	matrixStats_0.54.0
[19] bindr_0.1.1	plyr_1.8.4
[21] stringr_1.3.1	munsell_0.4.3
[23] gtable_0.2.0	rvest_0.3.3
[25] codetools_0.2-15	evaluate_0.10.1
[27] labeling_0.3	inline_0.3.15
[29] callr_3.0.0	ps_1.2.0
[31] parallel_3.5.0	highr_0.6
[33] Rcpp_0.12.18	readr_1.3.1
[35] scales_0.5.0	backports_1.1.2
[37] webshot_0.5.1	gridExtra_2.3
[39] hms_0.4.2	digest_0.6.15
[41] stringi_1.2.2	processx_3.2.0
[43] dplyr_0.7.6	grid_3.5.0
[45] rprojroot_1.3-2	cli_1.0.0
[47] tools_3.5.0	lazyeval_0.2.1
[49] tibble_1.4.2	crayon_1.3.4
[51] pkgconfig_2.0.2	xml2_1.2.0
[53] prettyunits_1.0.2	assertthat_0.2.0
[55] httr_1.4.0	rstudioapi_0.7
[57] R6_2.2.2	compiler_3.5.0