Logistic regression using Stan

Alessandro Varacca, Thomas Heckelei

July 2024

But what does logistic regression mean? (1)

$$y \sim \mathcal{D}(\boldsymbol{\theta})$$

- where $\boldsymbol{\theta}$ indicates a vector of parameters;
- When the outcome of some variable y is binary, $\mathcal{D} = \text{Ber}(\pi)$, where:

$$\text{Ber}(\pi) = \begin{cases} \pi & \text{if } y = 1, \\ 1 - \pi & \text{if } y = 0. \end{cases}$$

  and

$$\mathbb{E}[y] = \pi,$$
$$\mathbb{V}(y) = \pi(1 - \pi)$$

- In **logistic** regression models, we want to specify a functional from for $\mathbb{E}[y|X = x]$. However, unlike linear regression, this cannot be modelled **independently** of the the (conditional) variance $\mathbb{V}(y|X = x)$;

But what does logistic regression mean? (2)

- Since $\pi \in [0, 1]$, when we specify a functional form $\mathbb{E}[y|X = x]$ we need to make sure that is is constrained between 0 and 1;
- This is typically done via the so-called **link functions**:

$$\mathbb{E}[y|X = x] = g(\mu) = g\left(\alpha + \sum_{p=1}^{P} \beta_p x_p\right)$$

where either

$$g(\mu) = \begin{cases} \Phi(\mu) & \text{for Probit regression} \\ \Lambda(\mu) = \dfrac{1}{1 + \exp(\mu)} & \text{for Logistic regression.} \end{cases}$$

- We will use the second one.

How many parameters?

- The logistic regression model has $P+1$ parameters:
  1. α
  2. all the β_p
- We therefore need to set a prior on each of these parameters;
- To understand what kind of priors we need, it is important to understand what each of these parameters governs in the distribution on y_i;
- However, since y is binary, it is typically preferable to reason in terms of π, instead;
- Moreover, any change in α or β_p does not impact π_i directly, but it does so through the transformation $g(\alpha)$ or $g(\beta_p)$;
- In other words, $\beta_p = g^{-1}\{\mathbb{E}[y|X_p = x_p + 1]\} - g^{-1}\{\mathbb{E}[y|X_p = x_p]\}$ and, because of that, they are are hard to interpret on a probability scale;
- This makes prior calibration harder in logistic regression.

Calibrating the prior(s): α (1)

- We start off by studying a reasonable prior for α which, as in the linear regression case, is much easier to understand;
- Specifically, $\alpha = g^{-1}\{\mathbb{E}[y|X = 0]\} = g^{-1}\{\pi\}$;
- Therefore, to understand how a prior on α impacts π, we run the following simulation:
  1. Sample α^s from its prior;
  2. Calculate $\pi^s = g(\alpha^s)$
- Since this is relatively simple to implement, we immediatly try different configurations:

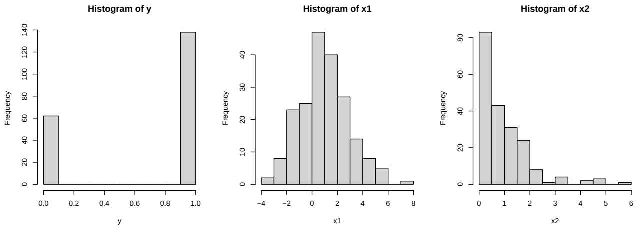
$$\alpha \sim \mathrm{N}(0, 10)$$
$$\alpha \sim \mathrm{N}(0, 2.5)$$
$$\alpha \sim \mathrm{N}(0, 1.5)$$
$$\alpha \sim \mathrm{N}(0, 1)$$

Load data and libraries

```r
library(rstan)
library(tidybayes)
library(tidyverse)
library(boot)

setwd("your working directory")

load("logistic_regression_data.RData")
source("aux_functions.R")

set.seed(123)
```

Visualize the data

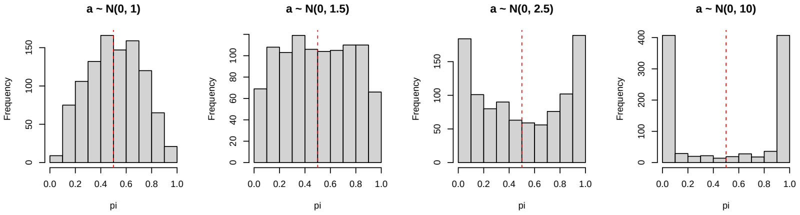
```
par(mfrow=c(1,3), pty="s")
hist(y)
hist(x1)
hist(x2)
```




```
par(mfrow=c(1,1), pty="m")
```

Calibrating the prior(s): α (2)

```
par(mfrow=c(1,4), pty="s")
for (j in c(1, 1.5, 2.5, 10)) {

  alpha <- rnorm(1000, 0, j)
  g_alpha <- inv.logit(alpha)
  hist(g_alpha, main = paste0("a ~ N(0, ", j, ")"), xlab = "pi")
  abline(v = 0.5, col = "red", lty = 2)

}
```




```
par(mfrow=c(1,1), pty="m")
```

Calibrating the prior(s): α (3)

- How to judge these priors?
- If we have no strong beliefs that π should be right, left skewed or skewed towards extreme ≈ 1 or ≈ 0 values, then $\mathrm{N}(0, 2.5)$ and $\mathrm{N}(0, 10)$ are **very** informative;
- In other words, choosing either of these distribution will yield priors placing a lot of emphasis on 'certain' zero or one values for y;
- On the other hand, **no information** in the context of binary outcomes corresponds to $\pi \approx 0.5$;
- Put it differently, before seeing the data, any observation should have 50-50 chance of being one (or zero);
- Therefore, $\alpha \sim \mathrm{N}(0, 1)$ is the most uninformative prior.

Calibrating β_1 and β_2 (1)

- Understanding what prior to give β_1 and β_2 is more complex;
- Not only do the distribution on these parameters will impact π, but so will their interaction with the covariates;
- As for α, let us see how different distribution choices for β_p play out: we begin by setting $\beta_1 = \beta_2 \sim \mathrm{N}(0, 5)$ and simulate:
  1. α^s from $\mathrm{N}(0, 1)$;
  2. β_1^s from $\mathrm{N}(0, 5)$;
  3. β_2^s from $\mathrm{N}(0, 5)$;
  4. $\pi_i^s = 1/(1 + \exp\{\alpha^s + \beta_1^s x_{i,1} + \beta_2^s x_{i,2}\})$.
- As in the linear regression case study, we run this simulation using a separate Stan program:

```
rstan_options(auto_write = TRUE)
logi_reg_prior <- stan_model("logistic_regression_prior.stan")
```

Calibrating β_1 and β_2 (2)

- As before, we must first set the data list:

```
dat_list <- list(
  N = length(y),
  x1 = x1,
  x2 = x2,
  y = y,
  mu_b1 = 0,
  mu_b2 = 0,
  sigma_b1 = 5,
  sigma_b2 = 5,
  mu_alpha = 0,
  sigma_alpha = 1
)
```

Calibrating β_1 and β_2 (3)

- We can now simulate $\tilde{\pi}^s$!

```
prior_sample <- sampling(logi_reg_prior,
                         data = dat_list,
                         algorithm = "Fixed_param")
```

- and extract the $N \times S$ matrix of simulated probability values:

```
prior_sample_fit_pi <- prior_sample %>% spread_draws(prob[condition])
```

- Notice that this look slightly different from the linear regression case study. We will see why in a second.

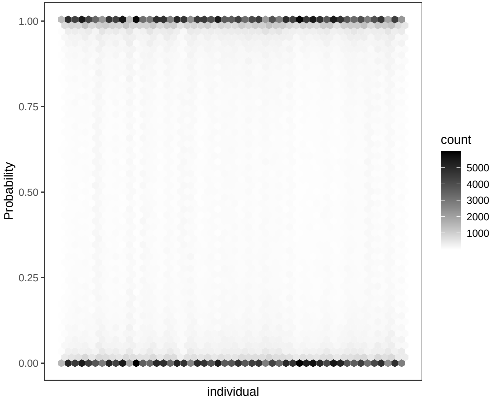
Calibrating β_1 and β_2 (4)

- Let us now visualize the simulated π_i^s;
- We can use the the functions in the `tidybayes` package;

```
prior_sample_fit_pi %>%
  ggplot(aes(x = condition, y = prob)) +
  geom_hex(bins = 50) +
  scale_fill_gradient(low = "white", high = "black") +
  theme_bw() +
  labs(x = "individual",
       y = "Probability") +
  theme(aspect.ratio = 1,
        axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
        panel.grid = element_blank())
```

Calibrating β_1 and β_2 (4)

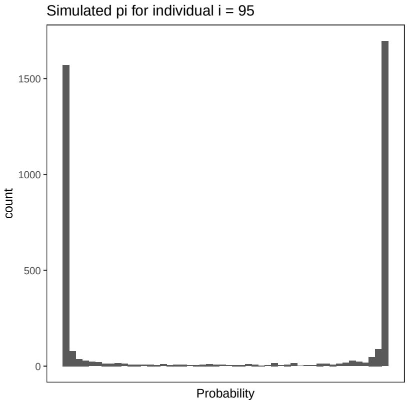
- Let us now visualize the simulated π_i^s;
- We can use the the functions in the `tidybayes` package;



- This plot is sort of a bird's-eye view of several sequential histograms.

Calibrating β_1 and β_2 (5)

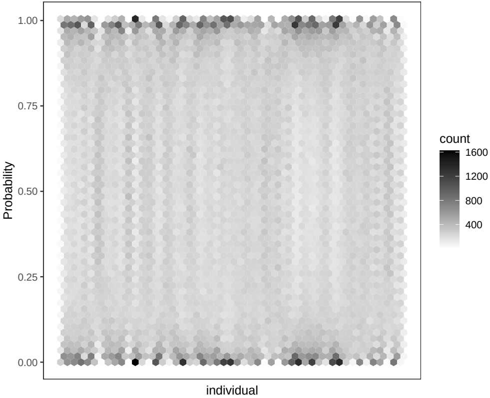
- For example, picking individual, say, $i = 95$ we have:



- Therefore, $\beta_1 = \beta_2 \sim N(0, 5)$ are very informative in that they automatically set probability values to their extremes.

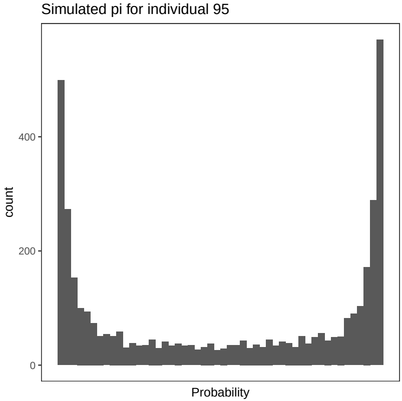
Calibrating β_1 and β_2 (5)

- What if we set $\beta_1 = \beta_2 \sim N(0, 1)$?



Calibrating β_1 and β_2 (6)

- For individual, say, $i = 95$ we have:



- Still very informative!
- So, what can we do if we want to keep prior information limited?

Default priors in logistic regression (1)

- One again, the prom is the scale-dependence of these coefficients;
- Even setting a relatively tight prior on β_p, when we multiply these coefficients by x_p, if the rage of the corresponding covariate is too big, this will have a disproportionate importance in pushing π to the extremes;
- One again, we can solve this issue by **standardizing**!
- In the context of logistic regression this practice is a bit less intuitive, although still fairly straightforward to carry out;
- The literature suggests that numeric variables should have mean zero and standard deviation one, while categorical variables should be centered on their mean:

$$\dot{z} = \begin{cases} (z - \hat{z})/\mathrm{sd}(z) * 0.5 & \text{if z is numeric,} \\ z - \bar{z} & \text{if z is binary.} \end{cases}$$

Default priors in logistic regression (2)

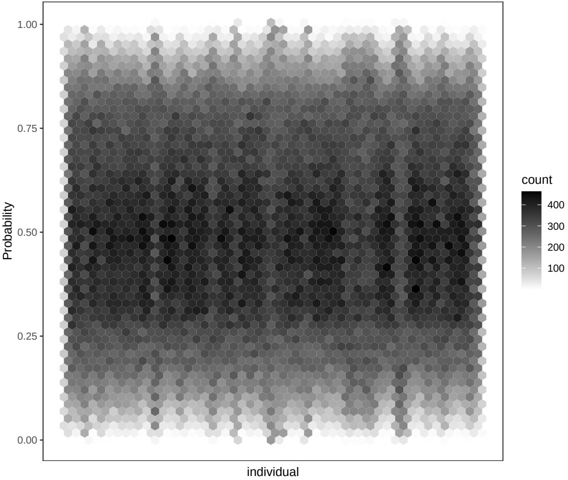
- Let us re-define our data list:

```
x1_sc <- as.vector(scale(x1)*0.5)
x2_sc <- as.vector(scale(x2)*0.5)

dat_list <- list(
  N = length(y),
  x1 = x1_sc,
  x2 = x2_sc,
  y = y,
  mu_b1 = 0,
  mu_b2 = 0,
  sigma_b1 = 1,
  sigma_b2 = 1,
  mu_alpha = 0,
  sigma_alpha = 1
)
```

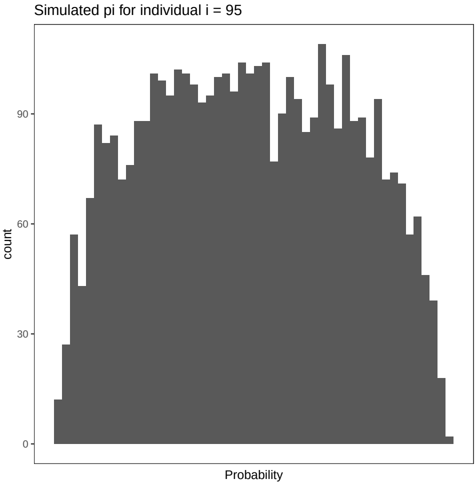
Default priors in logistic regression (3)

```
prior_sample <- sampling(logi_reg_prior,
                         data = dat_list,
                         algorithm = "Fixed_param")
prior_sample_fit_pi <- prior_sample %>% spread_draws(prob[condition])

prior_sample_fit_pi %>%
  ggplot(aes(x = condition, y = prob)) +
  geom_hex(bins = 50) +
  scale_fill_gradient(low = "white", high = "black") +
  theme_bw() +
  labs(x = "individual",
       y = "Probability") +
  theme(aspect.ratio = 1,
        axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
        panel.grid = element_blank())
```

Default priors in logistic regression (4)



Default priors in logistic regression (4)



Estimation

- Once all the priors have been setup, we can use `Stan` to sample from the implied joint posterior distribution:

$$f(\alpha, \beta_1, \beta_2 | \mathbf{y}, X_1 = x_1, X_2 = x_2)$$

- To do this, we need to load the second program:

```
logi_reg <- stan_model("logistic_regression.stan")
```

- Fitting the model to the data now only requesires a slighly different sampling statement:

```
fit <- sampling(logi_reg,
                data = dat_list,
                chains = 4,
                cores = 4)
sample_fit <- rstan::extract(fit)
```

Exploring the results (1)

To quickly visualize the results, we can call the function `summary` as we would do with the standard `lm` output:

```
params <- c("alpha", "beta1", "beta2")
summary(fit, pars = params)$summary[,1:3]
```

```
##             mean     se_mean        sd
## alpha  1.1697060 0.004120061 0.2003817
## beta1  3.1343891 0.009065092 0.4662018
## beta2 -0.7557881 0.006402164 0.3397158
```

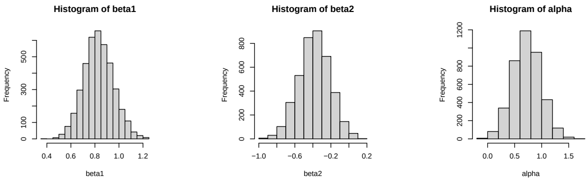
```
params <- c("alpha", "beta1", "beta2")
summary(fit, pars = params)$summary[,4:8]
```

```
##             2.5%        25%       50%        75%      97.5%
## alpha  0.7846771  1.0353133  1.166372  1.2978965  1.5599222
## beta1  2.2340592  2.8213259  3.119159  3.4400835  4.0990099
## beta2 -1.4233458 -0.9789671 -0.748847 -0.5217524 -0.1039808
```

Exploring the results (2)

- Because of standardization, we need to transform all the coefficients back to the original scale of the data;

```
beta1 <- sample_fit$beta1 * (0.5/sd(x1))
beta2 <- sample_fit$beta2 * (0.5/sd(x2))
alpha <- sample_fit$alpha - (beta1*mean(x1)) - (beta2*mean(x2))

par(mfrow=c(1,3), pty="s")
hist(beta1)
hist(beta2)
hist(alpha)
```




```
par(mfrow=c(1,1), pty="m")
```

Exploring the results (3)

```r
summary(beta1)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##  0.3958  0.7478  0.8268  0.8308  0.9119  1.2478
```

```r
summary(beta2)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## -0.9867 -0.4968 -0.3800 -0.3835 -0.2648  0.1220
```

```r
summary(alpha)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## -0.1763  0.5478  0.7287  0.7248  0.8963  1.7526
```

What if we used glm() instead?

```r
summary(glm(y ~ x1 + x2, family = binomial(link = "logit")))
```

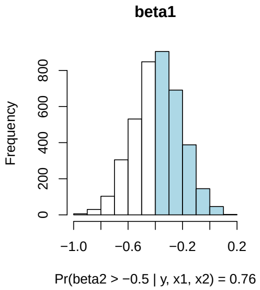
```
##
## Call:
## glm(formula = y ~ x1 + x2, family = binomial(link = "logit"))
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.7466     0.2886   2.587  0.00967 **
## x1            1.1288     0.1777   6.354  2.1e-10 ***
## x2           -0.4242     0.1968  -2.155  0.03116 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 247.64  on 199  degrees of freedom
## Residual deviance: 157.42  on 197  degrees of freedom
## AIC: 163.42
##
## Number of Fisher Scoring iterations: 6
```

Posterior probabilities (1)

- Just as in the linear regression case, we can calculate probabilities from the marginal posterior of the estimated parameters:

```
thr <- -0.5
prob <- mean(beta2>thr); prob
```

```
## [1] 0.75625
```



Credible Intervals

To calculate CrI, we simply take the corresponding quantile values:

```r
rbind(quantile(alpha, c(.025, .975)),
      quantile(beta1, c(.025, .975)),
      quantile(beta2, c(.025, .975)))
```

```
##            2.5%       97.5%
## [1,]  0.2142146  1.23708005
## [2,]  0.5921740  1.08650981
## [3,] -0.7222782 -0.05276515
```

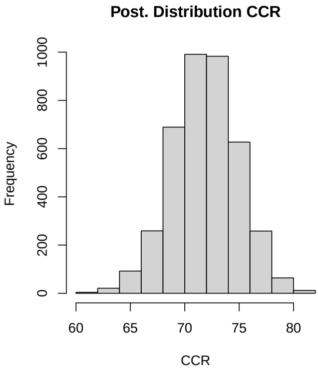
Model checking (internal consistency - 1)

- Unlike linear regression, we cannot evaluated goodness of fit by studying min, max and sd of the Posterior Predictive Distribution (PoPD);
- Inserade we need to fin a metric that make sense when using binary data;
- One such metric is the Correct Classification Rate (CCR), which indicates the proportion of observations correctly classified to either zero or one:

```
ccr <- numeric(length = nrow(sample_fit$y_pred))
for(s in 1:nrow(sample_fit$y_pred)) {

  conf_mat <- table(sample_fit$y_pred[s,], y)
  corr_class <- diag(conf_mat)

  ccr[s] <- (sum(corr_class) / sum(conf_mat)) * 100

}
```

Model checking (internal consistency - 2)

```
par(pty="s")
hist(ccr, main = "Post. Distribution CCR",
     xlab="CCR")
```




```
par(pty="m")
```