Spatial Cross-Valdiation

Adam Howes

June 2020

```
source("../../src/00_setup.R")

survey <- "MW2015DHS"

df <- readRDS("../../data/all.rds") %>%
  filter(survey_id == survey)
```

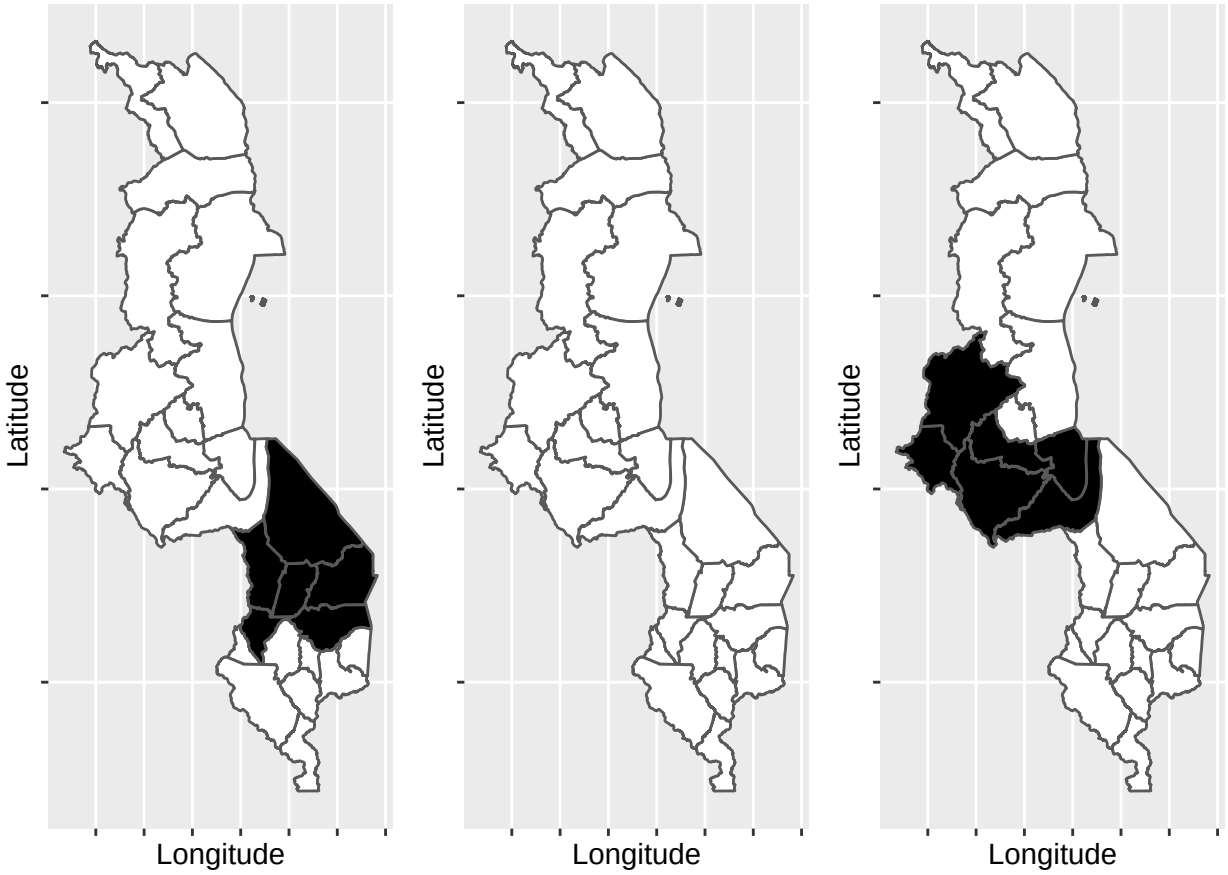
Function to produce Leave-One-Block-Out (LOBO) training sets, adapted from Eden. The input is `df`, a data matrix with response `y` and normalised response `est`. The ouput is a list of n dataframes, each with some of the responses replaced by `NA`.

```
lobo <- function(df){

  n <- nrow(df)
  nb <- poly2nb(df)
  training_sets <- vector(mode = "list", length = n)

  for(i in 1:nrow(df)) {
      df_new <- df

      # The neighbours of region i
      i_neighbours <- nb[[i]]
      held_out <- c(i, i_neighbours)

      # Replace the y and est entries by NA
      df_new[c(i, i_neighbours), c("y", "est")] <- rep(NA, length(i_neighbours) + 1)

      training_sets[[i]] <- list(data = df_new, centre = i, held_out = held_out)
  }

  return(training_sets)
}

training_sets <- lobo(df)
```

Example of three of the training sets:

```
plot_fold <- function(i) {
  training_sets[[i]]$data %>% # Balaka and surrounding districts removed
  ggplot(aes(fill = is.na(est))) +
  geom_sf(aes(geometry = geometry)) +
  coord_sf() +
  labs(x = "Longitude", y = "Latitude") +
  scale_fill_manual(values = c("white", "black")) +
  theme(axis.text = element_blank(),
        legend.position = "none")
}
```

```
cowplot::plot_grid(plot_fold(1), plot_fold(2), plot_fold(3), nrow = 1)
```



R-INLA

From the R-INLA FAQ:

> How can I do predictions using INLA? In INLA there is no function `predict` as for `glm`/`lm` in R. Predictions must to done as a part of the model fitting itself. As prediction is the same as fitting a model with some missing data, we can simply set `y[i] = NA` for those "locations" we want to predict.

SAE model with independent "spatial" random effects:

$$y_i \sim \text{Binomial}(m_i, \rho_i), \tag{1}$$
$$\text{logit}(\rho_i) = \beta_0 + \phi_i, \tag{2}$$
$$\beta_0 \sim p(\beta_0), \tag{3}$$
$$\phi_i \sim \mathcal{N}(0, \tau_\phi^{-1}), \quad i = 1, \ldots, n, \tag{4}$$
$$\tau_\phi \sim p(\tau_\phi). \tag{5}$$

Prior on precision τ_ϕ specified via $\sigma_\phi \sim \mathcal{N}(0, 2.5^2)$. The `initial` is in terms of $\theta = \log(\tau)$, so $\theta = 0$ corresponds to $\tau = 1$.

```
tau_prior <- list(prec = list(prior = "logtnormal", param = c(0, 1/2.5^2),
                              initial = 0, fixed = FALSE))
```

Code to fit:

```
inla_formula <- y ~ 1 + f(id, model = "iid", hyper = tau_prior)

inla_model <- function(set) {

  dat <- set$data

  inla_dat <- list(id = 1:nrow(dat), y = round(dat$y), m = dat$n_obs)

  fit <- inla(inla_formula,
              family = "binomial",
              control.family = list(control.link = list(model = "logit")),
              data = inla_dat,
              Ntrials = m,
              control.predictor = list(compute = TRUE, link = 1),
              control.compute = list(dic = TRUE, waic = TRUE,
                                     cpo = TRUE, config = TRUE))
  return(fit)
}

# Fit the model to each of the training sets
fitted_models <- lapply(training_sets, inla_model)
```

Some data structure management:

```
# Must be a better way to do this
# e.g. using mapply(function(x, y) append(x, list(fit = y)), training_sets, fitted_models)
# Premature optimisation!
n <- nrow(df)
cv <- vector(mode = "list", length = n)

for(i in 1:nrow(df)) {
  cv[[i]] <- append(training_sets[[i]], list(fit = fitted_models[[i]]))
}
```

Next step is to compare the training set fitted model predictions (in `fitted_model`) on the left out entries to the "correct answers".

Question A: what to compare to

What should be used as the correct answer? Some options are:

1. Fit a model (as below `full_model`) on all of the data and use that as the correct answer
2. Use the raw data

```
full_model <- inla_model(list(data = df))
```

Should the comparison be on the scale of

1. `df$y` the total cases
2. `df$rho` the prevalence

Using `df$y` will be unbalanced in favour of accuracy on larger regions. However, it could be argued that accuracy here is more important than treating every region more equally, as in `df$rho`.

Question B: where to predict

1. On all of the held-out entries
2. On only the central held-out entry (similar to LOO-CV)

Here I think that predicting on only the central held-out entry is preferable as

- Prediction on all held-out entries results in unequally repeated prediction accross folds (regions with more neighbours become more importatnt for the CV measure)
- The central region is most effective shielded from dependence with other regions

Question C: which loss function to use

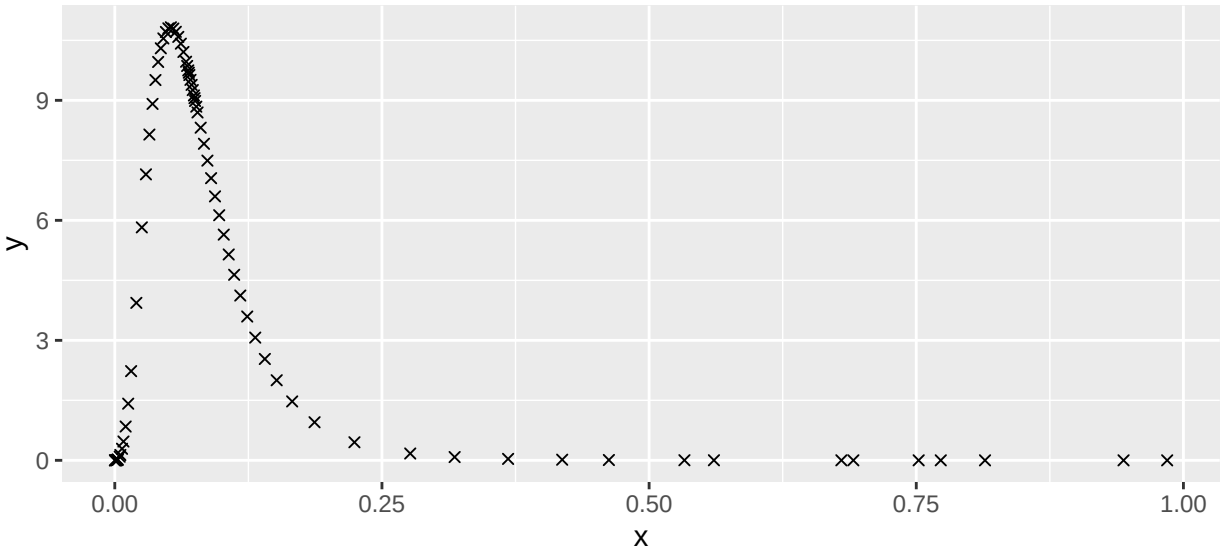
It seems unreasonable to apply a loss to a point estimate (scoring function) rather than try to use the posterior distribution (scoring rule). Consider a forecast (the posterior in this case) with probability density function f and cumulitive distribution function F and outcome z. Two options which Jeff has used before:

1. Log density score $\text{LDS}(f, z) = \log f(z)$
2. Continuous ranked probability score $\text{CRPS}(F, z) = -\int_{-\infty}^{\infty} (F(t) - I\{t \geq z\})^2 \, \mathrm{d}t$

```
# Test examples using the fitted model on the first fold
fit <- cv[[1]]$fit
```

The most informative thing to use, say for the central held-out entry in `fit`, is the following input-output pairs:

```
post_marginal <- fit$marginals.fitted.values$fitted.Predictor.01 %>% as.data.frame()

ggplot(post_marginal, aes(x = x, y = y)) +
  geom_point(shape = 4)
```



Interpolate for observed outcome z within the range defined using the `approx` function (`method = "linear"` by default). Possibly could use a kernel density smoother (this paper from 2011 suggests either the `ASH` or `KernSmooth` package) to create a higher resolution grid from which to interpolate. For example, to evaluate the posterior predictive density at `xout = 0.523` use:

```
approx(x = fit$marginals.fitted.values$fitted.Predictor.01[, 1],
       y = fit$marginals.fitted.values$fitted.Predictor.01[, 2],
       xout = 0.523)
```

```
## $x
```

```
## [1] 0.523
##
## $y
## [1] 0.003398534
```

Write a function to take an entry of `cv` and evaluate it's out-of-sample performance:

```
# I think this is the wrong approach, see later attempt

# evaluate_model <- function(model) {
#   i <- model$centre
#   marginal <- model$fit$marginals.fitted.values[[i]]
#   observed <- df$est[[i]]
#   density <- approx(x = marginal[, 1], y = marginal[, 2], xout = observed)
#   lds <- log(density$y) # The log density score
#   return(lds)
# }
#
# log_density_scores <- lapply(cv, evaluate_model) %>% unlist()
# sum(log_density_scores)
```

On questions A, B and C here I have chosen options (2, 1), 2 and 1 in this instance.

Notes from Monday morning HIV meeting

- Something about spatial CV that mightp be notable is that there are not new regions in the same way that there are new observations in other applications of modelling / CV
- Leaving blocks out is the gold-standard of spatial CV but may make prediction restrictively difficult causing all models to look very similar
  - Especially when there are no covariates (as in HIV)
- Leaving neighbours out may bias comparisons out of the favour of ICAR
- How often do you expect to have "huge" chunks of missing data?
- Is there an alternative approach based on sampling variability?
  - Leave-survey-out
  - Leave-"some proportion of the data"-out

Revisions

Question A was the primary place I was wrong or mislead. Fitting a model and using that as the training data is risky in that it supposes that the full model is right. Additionally, the predictive scores will no longer be interpretable (to the extent which they are interpretable in a spatial setting) in terms of "new" data. Moving on to `df$y` versus using `df$rho`. I wasn't clear that y_i are the data and ρ_i are parameters. In this instance, the predictive distribution of y_i is entirely determined by ρ_i (because m is fixed) though in a non-linear way. For this reason, although calibrating to ρ_i seems to achieve roughly the same objective it may not be advisable. Note that in simulation-based calibration comparisons are made on the parameters (unsure if this is relevant). As for the suggestion that using `df$y` could overweight the larger regions, this "trade-off" can be handled through the choice of loss function (additive or proportional).

Instead, use the following (Gelman, Hwang and Vehtari 2014; the paper has further detail and is worth reading in more depth) as an estimator of the log pointwise predictive density (LPPD):

$$\text{LPPD} \approx \sum_{i=1}^{n} \log \left(\frac{1}{S} \sum_{s=1}^{S} p\left(y_i \mid \theta^s\right) \right)$$

Where θ^s are the parameters. In this instance we will be replacing θ^s with ρ_i^s.

There is an issue here that may come up about simulation of the parameters. In the above equation θ^s is drawn from the full posterior distribution and so there is some dependence between predictions on each of the y_i. However, in INLA we only have access to the posterior marginal distributions. Skimming GHW 2014 I think that this issue comes up in Section 3.5 Pointwise vs. joint predictive distribution. This may not matter if you are only predicting on one held out entry (LOO) – each fold has a different posterior distribution.

Jeff suggests the function `INLA::inla.posterior.sample` (presumably performs implicit density estimation) in order to produce the samples.

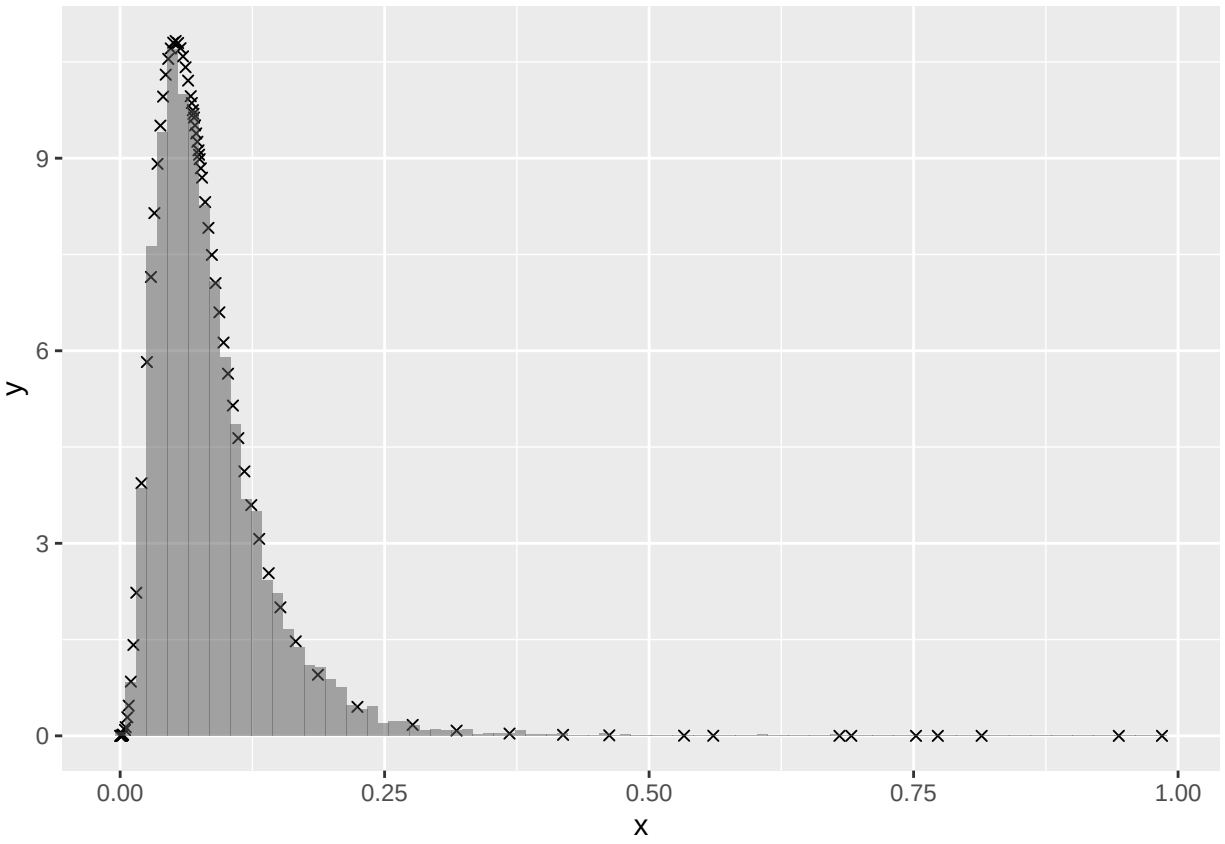
```
# Monte Carlo sample size
S <- 5000
```

Haakon (https://haakonbakkagit.github.io/btopic112.html) extracts the relevant parts of the samples after using `inla.posterior.sample`. However, there is an example of using the `selection` argument (https://becarioprecario.bitbucket.io/inla-gitbook/ch-INLA.html) which should be computationally more efficient.

```
# The linear predictors for i = 1
i <- 1
samples <- inla.posterior.sample(n = S, result = fit, selection = list(Predictor = i))

# Usually the latent component would include all n = 28 but not here
eta_samples = sapply(samples, function(x) x$latent)

# Transform to draws from rho
logistic <- function(eta) exp(eta) / (1 + exp(eta))
rho_samples <- logistic(eta_samples)
```

Demonstrating that the above procedure matches samples from the density implied by `post_marginal`:

```
ggplot() +
  geom_point(data = post_marginal,
             aes(x = x, y = y), shape = 4) +
  geom_histogram(data = as.data.frame(rho_samples),
                 aes(x = rho_samples, y = stat(100 * count / sum(count))),
                 bins = 100, alpha = 0.5)
```



The 1st entry $\log\left(\frac{1}{S}\sum_{s=1}^{S} p\left(y_1 \mid \theta^s\right)\right)$ of the LPPD is calculated by:

```
# The rounding issue coming up again
log(sum(dbinom(round(df$y[i]), df$n_obs[i], rho_samples)) / S)
```

```
## [1] -4.364528
```

So rewriting the `evaluate_model` function that previously was comparing values of ρ:

```
evaluate_model <- function(model, S = 5000) {
  i <- model$centre

  samples <- inla.posterior.sample(n = S, result = model$fit, selection = list(Predictor = i))
  eta_samples = sapply(samples, function(x) x$latent)
  rho_samples <- logistic(eta_samples)

  y <- df$y[[i]]
  n_obs <- df$n_obs[[i]]

  lds <- log(sum(dbinom(round(df$y[i]), df$n_obs[i], rho_samples)) / S)

  return(lds)
}

# Looks within MC error of the above version
evaluate_model(cv[[1]], S = 1000)
```

```
## [1] -4.378501
```

Apply it over all 28 of the cross-validation sets in `cv`:

```r
log_density_scores <- lapply(cv, evaluate_model) %>% unlist()
sum(log_density_scores)
```

```
## [1] -128.1081
```

Putting it into operation

- Lots of reformatting
- I first tried replicating the results I got on this file with my full code. For the 21st region of Malawi being the central held-out for some reason `inla.posterior.sample` gives the error: `Error in inla.posterior.sample.interpret.selection(selection, result): all(sel <= len) is not TRUE`. Currently working on resolving this.