A short tutorial on fitting the models in R

1 Data

1.1 Tidyverse and example data

For the most part of this exercise, we use the `tidyverse` packages. We use simulated dataset based loosely on Nigeria: 200 clusters from MIS 2015 and 200 clusters from MIS 2010. Furthermore, the dataset is already divided into a test set (20 randomly chosen clusters out of 200 clusters in 2015) and a training set (the remaining 180 clusters in 2015 and 200 clusters in 2010).

```
library(tidyverse, quietly = T)
library(sp)
library(raster)
```

```
traindat <- read_csv("https://raw.github.com/kokbent/mal_mod_comp/master/traindat.csv")
```

```
## Parsed with column specification:
## cols(
##   cluster = col_double(),
##   year = col_double(),
##   malaria = col_double(),
##   temp = col_double(),
##   prec = col_double(),
##   lng = col_double(),
##   lat = col_double()
## )
```

```
testdat <- read_csv("https://raw.github.com/kokbent/mal_mod_comp/master/testdat.csv")
```

```
## Parsed with column specification:
## cols(
##   cluster = col_double(),
##   year = col_double(),
##   malaria = col_double(),
##   temp = col_double(),
##   prec = col_double(),
##   lng = col_double(),
##   lat = col_double()
## )
```

```
head(traindat)
```

```
## # A tibble: 6 x 7
##   cluster  year malaria   temp   prec   lng   lat
##     <dbl> <dbl>   <dbl>  <dbl>  <dbl> <dbl> <dbl>
## 1       1     0       1 -0.994 -0.536  6.51  5.89
## 2       1     0       0 -0.994 -0.536  6.51  5.89
## 3       1     0       0 -0.994 -0.536  6.51  5.89
## 4       1     0       0 -0.994 -0.536  6.51  5.89
## 5       1     0       0 -0.994 -0.536  6.51  5.89
## 6       1     0       0 -0.994 -0.536  6.51  5.89
```

The data contains information of every individual that was sampled for malaria test. The columns of the dataframe are:

- `cluster`: The cluster ID of the individual
- `year`: Dummy variable for the sampling year, 0 for "past" data (MIS 2010) and 1 for "present" data (MIS 2015).
- `malaria`: Malaria microscopy test outcome, 0 negative, 1 positive
- `temp`: Standardized average land surface temperature
- `prec`: standardized average precepitation (rainfall)
- `lng`: Longitude (based on WGS84)
- `lat`: Latitude (based on WGS84)

Although we found no clear difference in using projected vs unprojected coordinates systems in the models, here we project the longitude and latitude to x- and y-coordinates:

```
xy_train <- traindat[, c("lng", "lat")] %>%
  SpatialPoints(proj4string = CRS("+init=epsg:4326")) %>%
  spTransform(CRS("+init=epsg:32632")) %>%
  coordinates()
traindat$lng <- xy_train[,"lng"]
traindat$lat <- xy_train[,"lat"]

xy_test <- testdat[, c("lng", "lat")] %>%
  SpatialPoints(proj4string = CRS("+init=epsg:4326")) %>%
  spTransform(CRS("+init=epsg:32632")) %>%
  coordinates()
testdat$lng <- xy_test[,"lng"]
testdat$lat <- xy_test[,"lat"]

head(traindat)
```

```
## # A tibble: 6 x 7
##   cluster  year malaria   temp   prec     lng     lat
##     <dbl> <dbl>   <dbl>  <dbl>  <dbl>   <dbl>   <dbl>
## 1       1     0       1 -0.994 -0.536 224410. 651338.
## 2       1     0       0 -0.994 -0.536 224410. 651338.
## 3       1     0       0 -0.994 -0.536 224410. 651338.
## 4       1     0       0 -0.994 -0.536 224410. 651338.
## 5       1     0       0 -0.994 -0.536 224410. 651338.
## 6       1     0       0 -0.994 -0.536 224410. 651338.
```

```
head(testdat)
```

```
## # A tibble: 6 x 7
##   cluster  year malaria  temp  prec     lng     lat
##     <dbl> <dbl>   <dbl> <dbl> <dbl>   <dbl>   <dbl>
## 1      10     1       1 -1.33 0.325 145473. 633496.
## 2      10     1       1 -1.33 0.325 145473. 633496.
## 3      10     1       1 -1.33 0.325 145473. 633496.
## 4      10     1       1 -1.33 0.325 145473. 633496.
## 5      10     1       1 -1.33 0.325 145473. 633496.
## 6      10     1       1 -1.33 0.325 145473. 633496.
```

2 Model fitting & prediction

In this section, we demonstrate how we fit each of the five types of models described in the main article to the training dataset and use these models to predict malaria prevalence in the test dataset. For all models, the prediction will be based on location (cluster) aggregated data.

```
test_locdat <- testdat %>%
  group_by(cluster) %>%
  mutate(n = n(), m = sum(malaria)) %>%
  dplyr::select(-malaria) %>%
  distinct()
head(test_locdat)
```

```
## # A tibble: 6 x 8
## # Groups:   cluster [6]
##   cluster  year   temp    prec     lng      lat     n     m
##     <dbl> <dbl>  <dbl>   <dbl>   <dbl>    <dbl> <int> <dbl>
## 1      10     1 -1.33   0.325  145473.  633496.    27     6
## 2      14     1  1.62  -0.419  341344. 1233619.    23     8
## 3      26     1 -0.722  0.734  718629. 1368156.    28     8
## 4      28     1 -1.45  -0.0671 149324. 1131975.    30     1
## 5      42     1  1.25   0.527  625653. 1376535.    21    14
## 6      46     1  0.995  0.447  614006. 1124484.    24     9
```

Column `n` here contains the overall number of children in each cluster while the column `m` contains the number of children in each cluster that were tested positive for malaria.

We then store the predicted prevalence of each location by each model and setting (spatial-only or spatiotemporal) combination in a matrix `prediction`. We pre-name the columns here, note that "S" refers to Spatial-only setting, and "ST" refers to Spatiotemporal setting.

```
# Creating empty 'prediction' matrix to store model predicted prevalence
# There are four models x two settings = 8 columns
prediction <- matrix(NA, nrow(test_locdat), 8)

colnames(prediction) <- c("StepGLM-S", "StepGLM-ST",
                          "GAM-S", "GAM-ST",
                          "GBM-S", "GBM-ST",
                          "SPDE-INLA-S", "SPDE-INLA-ST")
```

In Section 3, we calculate the predictive performance of the model-setting from the `prediction` matrix using two metrics: the mean loglikelihood (`logLik`) and weighted Mean Absolute Error (`MAE`).

Note that under the spatial setting, we use only the "present" or the newest dataset to train the model. As a result, we filter the training dataset (e.g. `traindat$year == 1`) under this setting. Under the spatio-temporal setting, the full dataset is used to train the model, so filtering does not occur.

2.1 Stepwise logistic regression

2.1.1 Spatial Setting

The model is fit using the `glm()` function and stepwise selection using AIC is conducted using the `stepAIC()` from the `MASS` package.

```r
library(MASS)
```

```r
f <- malaria ~ temp + prec + lng + lat
mod <- glm(f, family = binomial(), data = traindat, subset = (traindat$year == 1))
mod <- stepAIC(mod, trace = F)
```

We can obtain the predicted probability (prevalence) using the `predict()` function in which `type = "response"`. We store the predicted prevalence in the `prediction` matrix, and will calculate the predictive performance measure after collecting predicted prevalences from all model-setting in Section 3.

```r
# Obtaining prediction
prediction[,1] <- predict(mod, test_locdat, type = "response")
```

2.1.2 Spatio-temporal

The spatio-temporal setting is fit similarly to the spatial setting. We add the `year` dummy variable into the formula, and the `subset` argument in the `glm()` is not specified.

```r
f <- malaria ~ temp + prec + lng + lat + year
mod <- glm(f, family = binomial(), data = traindat)
mod <- stepAIC(mod, trace = F)
```

The predictions are obtained similarly to that of spatial setting:

```r
# Obtaining prediction
prediction[,2] <- predict(mod, test_locdat, type = "response")
```

2.2 Generalized Additive Model (GAM)

2.2.1 Spatial Setting

We use `mgcv` package to fit GAM with a spatial smoother (2D splines involving x- and y-coordinates):

```r
library(mgcv)
```

```
f <- malaria ~ temp + prec + lng + lat + s(lng, lat, bs="tp")
mod <- gam(f, family = binomial(), data = traindat, subset = traindat$year == 1,
           method = "REML")
```

Similar to the logistic regression, obtaining the predicted probability (prevalence) of each cluster is straightforward:

```
# Obtaining prediction
prediction[,3] <- predict(mod, test_locdat, type = "response")
```

2.2.2 Spatio-temporal setting

In this setting, we apply spatiotemporal smoother involving x-, y-coordinates and year.

```
f <- malaria ~ temp + prec + lng + lat + year + s(lng, lat, year, bs="tp")
mod <- gam(f, family = binomial(), data = traindat,
           method = "REML")

# Obtaining prediction
prediction[,4] <- predict(mod, test_locdat, type = "response")
```

2.3 Gradient Boosted Trees

2.3.1 Spatial setting

For gradient boosted trees, we use the `gbm` package to fit the model. We use the `caret` package to conduct cross validation to find the optimal values for the parameters. These parameters are maximum depth for each tree (d): `interaction.depth`; total number of trees (B): `n.trees`; shrinkage (*lambda*): `shrinkage`. Specifically, we created a grid of parameters and use the `train()` function to search the best combination of the three parameters. We found that the model performance was generally insensitive to the choice of minimum number of observations per terminal nodes (`n.minobsinnode`) and we use the default choice of 10.

```
library(caret)
library(gbm)
```

```
f <- as.factor(malaria) ~ temp + prec + lng + lat
grid <- expand.grid(interaction.depth = 1:5, n.trees = 1:3 * 100,
                    shrinkage = 10^(-2:1), n.minobsinnode = 10)
fitControl <- trainControl(method = "cv", number = 5)
mod <- train(f, data = traindat, method = "gbm", trControl = fitControl,
             subset = traindat$year == 1, train.fraction = 0.75,
             verbose = F, tuneGrid = grid)
```

The “best” parameters are as follows:

```
mod$bestTune
```

```
##   n.trees interaction.depth shrinkage n.minobsinnode
## 9     300                 3      0.01             10
```

As with all other models, prediction is straightforward with the `predict()` function, although under the `gbm` package, it returns the probability of both 0 and 1 so we need to specify that we want the probability of 1:

```
# Obtaining prediction
pred <- predict(mod, test_locdat, type = "prob")
prediction[,5] <- pred$`1`
```

2.3.2 Spatio-temporal setting

In this setting, we add the `year` dummy variable to the formula while all other steps are identical to that of the spatial setting.

```
f <- as.factor(malaria) ~ temp + prec + lng + lat + year
grid <- expand.grid(interaction.depth = 1:5, n.trees = 1:3 * 100,
                    shrinkage = 10^(-2:1), n.minobsinnode = 10)
fitControl <- trainControl(method = "cv", number = 5)
mod <- train(f, data = traindat, method = "gbm", trControl = fitControl,
             train.fraction = 0.75, verbose = F, tuneGrid = grid)
```

```
# Obtaining prediction
pred <- predict(mod, test_locdat, type = "prob")
prediction[,6] <- pred$`1`
```

2.4 SPDE-INLA

2.4.1 Spatial-Setting

We use the INLA package available in http://www.r-inla.org/download. Details regarding how to download and install this package can also be found in this website. We also use the `sp` package to convert the coordinates to `SpatialPoints` object, and the `raster` package to visualize the mesh and the country's shape file.

```
library(INLA)
```

Unlike other packages, `INLA` is fitted on aggregated data. Therefore, the original training dataset is transformed to an aggregated dataset `train_locdat` here. Additionally, we create matrices of coordinates for training and test sets.

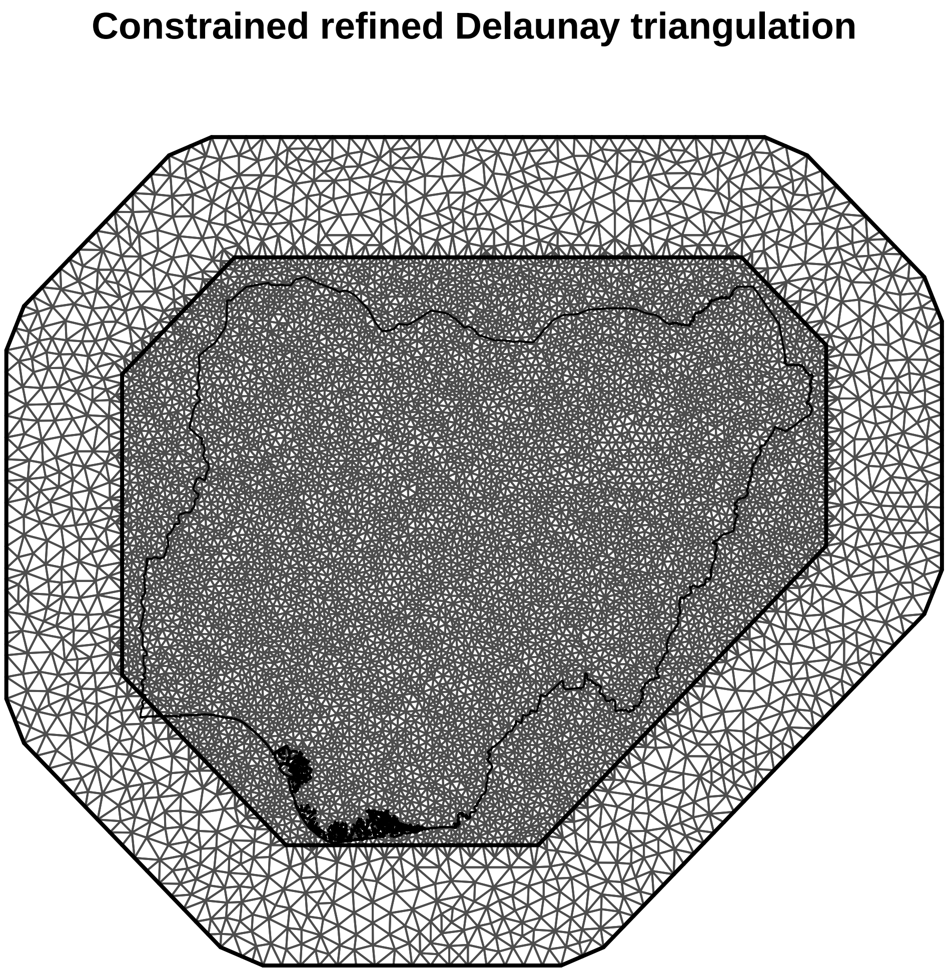
```
train_locdat <- traindat %>%
  filter(year == 1) %>%
  group_by(cluster) %>%
  mutate(n = n(), m = sum(malaria)) %>%
  dplyr::select(-malaria) %>%
  distinct()

coords_train <- train_locdat[train_locdat$year == 1, c("lng", "lat")] %>%
  as.data.frame() %>% as.matrix()
coords_hold <- test_locdat[, c("lng", "lat")] %>%
  as.data.frame() %>% as.matrix()

#coords_train <- SpatialPoints(train_locdat[train_locdat$year == 1, c("lng", "lat")],
#                               proj4string = CRS("+init=epsg:4326"))
#coords_hold <- SpatialPoints(test_locdat[, c("lng", "lat")],
#                               proj4string = CRS("+init=epsg:4326"))
```

We use the `inla.mesh.2d()` function to create a triangular mesh based on the training coordinates. The mesh size is controlled by several parameters and here we chose to specify the `cutoff` and `max.edge` parameters. `cutoff` refers to the minimum allowed distance between points; we chose a reasonably small value (500) here so that the clusters (locations) with data are not lumped together. Note that this correspond to 500 meters since we are using projected coordinate system here. `max.edge` refers to the largest allowed triangle edge length: when given one value, the function creates a mesh only up to the extent of the coordinates; when given two values, the function adds a "buffer" outside of the extent of the coordinates, which is particularly important here because there's no guarantee that the "inner" mesh would completely cover the country. It is important to note that a tradeoff exists between computational effort and coarseness of predictions when choosing the `max.edge`. Smaller values, lead to a finer mesh size and more triangles, and more time is required to fit the model. In our setting, we found that 20000 for inner mesh was optimal, resulting in a mesh of 9258 triangles: fine enough for good approximation and small enough that it doesn't take too long to fit.

```
# The shapefiles of Nigeria is only required to visualize the mesh coverage with
# respect to the country.
download.file("https://raw.github.com/kokbent/mal_mod_comp/master/NGA_adm0.rds",
              "NGA_adm0.rds", mode = "wb" )

NG_shp <- read_rds("NGA_adm0.rds")

# Project the shape to UTM
NG_shp <- NG_shp %>%
  spTransform(CRS("+init=epsg:32632"))

mesh <- inla.mesh.2d(coords_train,
                     cutoff = 500,
                     max.edge = c(20000, 50000))
mesh$n
```

```
## [1] 9258
```

```
plot(mesh)
plot(NG_shp, add=T)
```



Note that the inner mesh appears as a shaded gray region here because the mesh is very fine (i.e., individual triangles are hard to distinguish because they are very small). The inner mesh can be better visualized if the plot is enlarged.

In this simulated dataset, the coverage of the mesh is almost perfect in that the whole country falls within the inner mesh. However, when using real dataset, some parts of the country may fall out of the inner mesh, which is inevitable when no cluster was sampled in these areas. We then create a Matern SPDE model object:

```
spde <- inla.spde2.matern(mesh=mesh, alpha=2)
```

To "map" the sampled locations to the mesh, we construct the observation weight matrix using the `inla.spde.make.A()` function:

```r
A <- inla.spde.make.A(mesh, loc=coords_train)
```

Here we "stack" the input data, response and the observation matrix using the `inla.stack()` function. The input data is different from that of other models. Specifically, INLA requires aggregated data, where the "response" variable for each cluster is the number of individuals infected with malaria (`m`) and the total number of individuals sampled in each cluster (`n`):

```r
fieldInd <- inla.spde.make.index("field", n.spde=mesh$n)

f <- ~ temp + prec + lng + lat - 1
cov_df <- model.matrix(f, train_locdat) %>% as.data.frame()
cov_df <- cbind(Intercept = 1, cov_df, n = train_locdat$n)

stk.dat <- inla.stack(data=list(y=train_locdat$m),
                      A=list(A,1), tag='dat',
                      effects=list(fieldInd, cov_df))
```

Similarly, the prediction weight matrix is created and "stacked" with the covariates of heldout data:

```r
Aprd <- inla.spde.make.A(mesh, loc=coords_hold)

cov_df <- model.matrix(f, test_locdat) %>% as.data.frame()
cov_df <- cbind(Intercept = 1, cov_df, n = NA)

stk.prd <- inla.stack(data=list(y=NA),
                      A=list(Aprd, 1), tag='prd', effects=list(fieldInd, cov_df))
```

Then we stack the "observation" stack with the "prediction" stack:

```r
stk.all <- inla.stack(stk.dat, stk.prd, remove.unused = T)
N <- c(train_locdat$n, rep(NA, nrow(coords_hold)))
```

We specify the formula for the SPDE-INLA model, and fit it using the `INLA()` function. Both model fitting and prediction are conducted here:

```r
f <- y ~ 0 + Intercept + lng + lat + temp + prec + f(field, model=spde)
mod <- inla(f, family='binomial',
           control.compute=list(cpo=TRUE),
           data=inla.stack.data(stk.all),
           control.predictor=list(
             A=inla.stack.A(stk.all), compute=T, link=1),
           control.inla= list(strategy = 'gaussian',
                              int.strategy='eb',
                              fast=TRUE,dz=1,
                              step.factor=0.5,
                              stupid.search=FALSE),
           Ntrials = N)
```

The prediction points are tagged with `prd`, and we can retrieve the posterior mean predicted probability:

```r
# Obtaining prediction
id.prd <- inla.stack.index(stk.all, 'prd')$data

prediction[,7] <- mod$summary.fitted.values$mean[id.prd]
```

2.4.2 Spatiotemporal setting

The first steps are similar to the spatial settings. However, the number of triangle in the mesh is larger due to the increased observations (clusters), which often results in an inner mesh with better coverage throughout the country.

```r
train_locdat <- traindat %>%
  group_by(cluster) %>%
  mutate(n = n(), m = sum(malaria)) %>%
  dplyr::select(-malaria) %>%
  distinct()

coords_train <- train_locdat[, c("lng", "lat")] %>%
  as.data.frame() %>% as.matrix()
coords_hold <- test_locdat[, c("lng", "lat")] %>%
  as.data.frame() %>% as.matrix()

mesh <- inla.mesh.2d(coords_train,
                     cutoff = 500,
                     max.edge = c(20000, 50000))
mesh$n
```

```
## [1] 9540
```

```r
spde <- inla.spde2.matern(mesh=mesh, alpha=2)
```

In the spatio-temporal setting, we also create a 1D "mesh" that corresponds to the temporal component:

```r
mesh1d <- inla.mesh.1d(0:1, boundary=c('free'))
```

Besides mapping the cluster coordinates onto the 2D mesh, the sampling year of the cluster is also mapped to the 1D mesh:

```r
A <- inla.spde.make.A(mesh, loc=coords_train, group = train_locdat$year, group.mesh = mesh1d)
```

Since the observation matrix's size is doubled (due to the additional temporal factor), the index vector `fieldInd` made using the `inla.spde.make.index()` function is doubled with additional argument. The input data is then stacked with the observation matrix:

```r
fieldInd <- inla.spde.make.index("field", n.spde=mesh$n, n.group=mesh1d$m)

f <- ~ temp + prec + lng + lat + year - 1
cov_df <- model.matrix(f, train_locdat) %>% as.data.frame()
cov_df <- cbind(Intercept = 1, cov_df, n = train_locdat$n)
```

```
stk.dat <- inla.stack(data=list(y=train_locdat$m),
                      A=list(A,1), tag='dat',
                      effects=list(fieldInd, cov_df))
```

Similarly, the prediction weight matrix is created and "stacked" with the covariates of heldout data. Then we stack the observation and prediction stacks:

```
Aprd <- inla.spde.make.A(mesh, loc=coords_hold, group = test_locdat$year, group.mesh = mesh1d)

cov_df <- model.matrix(f, test_locdat) %>% as.data.frame()
cov_df <- cbind(Intercept = 1, cov_df, n = NA)

stk.prd <- inla.stack(data=list(y=NA),
                      A=list(Aprd, 1), tag='prd', effects=list(fieldInd, cov_df))

stk.all <- inla.stack(stk.dat, stk.prd, remove.unused = T)
N <- c(train_locdat$n, rep(NA, nrow(coords_hold)))
```

The formula for the Spatiotemporal model is slightly different. Not only the `year` dummy covariate is entered, additional arguments are added to `f(field, model=spde)` to specify the temporal component, and its underlying autoregressive lag-1 (AR1) model.

```
f <- y ~ 0 + Intercept + lng + lat + temp + prec + year +
  f(field, model=spde, group = field.group, control.group=list(model='ar1'))
mod <- inla(f, family='binomial',
           control.compute=list(cpo=TRUE),
           data=inla.stack.data(stk.all),
           control.predictor=list(
             A=inla.stack.A(stk.all), compute=T, link=1),
           control.inla= list(strategy = 'gaussian',
                              int.strategy='eb',
                              fast=TRUE,dz=1,
                              step.factor=0.5,
                              stupid.search=FALSE),
           Ntrials = N)
```

Prediction is similar to that of the spatial settings:

```
# Obtaining prediction
id.prd <- inla.stack.index(stk.all, 'prd')$data
prediction[,8] <- mod$summary.fitted.values$mean[id.prd]
```

3. Model comparisons

Through the Section 2, we now have ten sets of predicted probabilities in the `prediction` matrix, and we can use these values to calculate each model-setting's predictive performance.

```
colnames(prediction) <- c("StepGLM-S", "StepGLM-ST",
                          "GAM-S", "GAM-ST",
                          "GBM-S", "GBM-ST",
                          "SPDE-INLA-S", "SPDE-INLA-ST")
round(head(prediction), 2)
```

```
##      StepGLM-S StepGLM-ST GAM-S GAM-ST GBM-S GBM-ST SPDE-INLA-S SPDE-INLA-ST
## [1,]      0.09       0.12  0.10   0.11  0.17   0.13        0.09         0.27
## [2,]      0.23       0.19  0.25   0.21  0.12   0.13        0.24         0.36
## [3,]      0.31       0.29  0.30   0.28  0.36   0.39        0.30         0.44
## [4,]      0.04       0.04  0.04   0.04  0.12   0.04        0.04         0.18
## [5,]      0.58       0.54  0.58   0.55  0.46   0.44        0.63         0.53
## [6,]      0.50       0.50  0.50   0.52  0.41   0.59        0.51         0.50
```

We use the `dbinom()` function to calculate the per-person log-likelihood (`logLik`). Higher `logLik`, i.e. closer to zero, is better. We first calculate the log-likelihood of each cluster, then we sum the log-likehood of all clusters and divide by the total number of individuals:

```
LL <- apply(prediction, 2,
            function (p) dbinom(x = test_locdat$m, size = test_locdat$n, prob = p, log = T))
LL <- colSums(LL) / sum(test_locdat$n)
LL
```

```
##    StepGLM-S   StepGLM-ST        GAM-S       GAM-ST        GBM-S       GBM-ST
##  -0.09725356  -0.09494780  -0.09340545  -0.09653973  -0.10277721  -0.11390857
##  SPDE-INLA-S SPDE-INLA-ST
##  -0.09748326  -0.17160939
```

For the weighted Mean Absolute Error (`MAE`), we first calculate the AE or absolute difference in predicted vs actual prevalence in each cluster. Then we weight the AEs by sample size (number of individuals tested) in each cluster to obtain the MAE. Lower `MAE` is better:

```
AE <- apply(prediction, 2, function (p) abs((test_locdat$m / test_locdat$n) - p))
MAE <- apply(AE, 2, function (ae) sum(test_locdat$n * ae) / sum(test_locdat$n))
MAE
```

```
##    StepGLM-S   StepGLM-ST        GAM-S       GAM-ST        GBM-S       GBM-ST
##   0.08271344   0.07845092   0.07805452   0.08082387   0.08804125   0.10712786
##  SPDE-INLA-S SPDE-INLA-ST
##   0.08267044   0.16139691
```

Based on the `logLik` & `MAE`, we would conclude that GAM under spatial setting is the best model in this exercise. Note that the findings in this tutorial, which uses a simulated dataset, are different from that of the main article which uses real world dataset.