Package ‘disaggregation’

September 7, 2023

Type Package

Title Disaggregation Modelling

Version 0.2.1

Description Fits disaggregation regression models using 'TMB' ('Template Model Builder'). When the response data are aggregated to polygon level but the predictor variables are at a higher resolution, these models can be useful. Regression models with spatial random fields. The package is described in detail in Nandi et al. (2023) <doi:10.18637/jss.v106.i11>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

Imports raster, foreach, sp, sf, parallel, doParallel, splancs, Matrix, stats, TMB, dplyr, ggplot2, cowplot, sparseMVN, utils

Additional_repositories `https://inla.r-inla-download.org/R/stable`

Suggests testthat, INLA, knitr, rmarkdown, SpatialEpi

LinkingTo TMB, RcppEigen

SystemRequirements GNU make

VignetteBuilder knitr

NeedsCompilation yes

Author Anita Nandi [aut] (<`https://orcid.org/0000-0002-5087-2494`>),
Tim Lucas [aut, cre] (<`https://orcid.org/0000-0003-4694-8107`>),
Rohan Arambepola [aut],
Andre Python [aut] (<`https://orcid.org/0000-0001-8094-7226`>)

Maintainer Tim Lucas <`timcdlucas@gmail.com`>

Repository CRAN

Date/Publication 2023-09-07 21:40:05 UTC

R topics documented:

as.disag_data . . . 2
build_mesh . . . 4
disaggregation-deprecated . . . 5
dummy . . . 5
fit_model . . . 6
getCovariateRasters . . . 9
getPolygonData . . . 10
getStartendindex . . . 11
make_model_object . . . 12
parallelExtract . . . 15
plot.disag_data . . . 16
plot.disag_model . . . 17
plot.disag_prediction . . . 17
predict.disag_model . . . 18
predict_model . . . 19
predict_uncertainty . . . 20
prepare_data . . . 21
print.disag_data . . . 24
print.disag_model . . . 25
print.disag_prediction . . . 25
summary.disag_data . . . 26
summary.disag_model . . . 26
summary.disag_prediction . . . 27

Index **28**

`as.disag_data` *Function to fit the disaggregation model*

Description

Function to fit the disaggregation model

Usage

```
as.disag_data(
  polygon_shapefile,
  shapefile_names,
  covariate_rasters,
  polygon_data,
  covariate_data,
  aggregation_pixels,
  coordsForFit,
  coordsForPrediction,
  startendindex,
  mesh = NULL
)
```

Arguments

`polygon_shapefile`
: SpatialPolygonDataFrame containing the response data

`shapefile_names`
: List of 2: polygon id variable name and response variable name from polygon_shapefile

`covariate_rasters`
: RasterStack of covariates

`polygon_data`
: data.frame with two columns: polygon id and response

`covariate_data`
: data.frame with cell id, polygon id and covariate columns

`aggregation_pixels`
: vector with value of aggregation raster at each pixel

`coordsForFit`
: coordinates of the covariate data points within the polygons in polygon_shapefile

`coordsForPrediction`
: coordinates of the covariate data points in the whole raster extent

`startendindex`
: matrix containing the start and end index for each polygon

`mesh`
: inla.mesh object to use in the fit

Value

A list is returned of class `disag_data`. The functions *summary*, *print* and *plot* can be used on `disag_data`. The list of class `disag_data` contains:

`polygon_shapefile`
: The SpatialPolygonDataFrame used as an input.

`covariate_rasters`
: The RasterStack used as an input.

`polygon_data`
: A data frame with columns of *area_id*, *response* and *N* (sample size: all NAs unless using binomial data). Each row represents a polygon.

`covariate_data`
: A data frame with columns of *area_id*, *cell_id* and one for each covariate in *covariate_rasters*. Each row represents a pixel in a polygon.

`aggregation_pixels`
: An array with the value of the aggregation raster for each pixel in the same order as the rows of *covariate_data*.

`coordsForFit`
: A matrix with two columns of x, y coordinates of pixels within the polygons. Used to make the spatial field.

`coordsForPrediction`
: A matrix with two columns of x, y coordinates of pixels in the whole Raster. Used to make predictions.

`startendindex`
: A matrix with two columns containing the start and end index of the pixels within each polygon.

`mesh`
: A INLA mesh to be used for the spatial field of the disaggregation model.

build_mesh *Build mesh for disaggregaton model*

Description

build_mesh function takes a SpatialPolygons object and mesh arguments to build an appropriate mesh for the spatial field.

Usage

```
build_mesh(shapes, mesh.args = NULL)
```

Arguments

`shapes`	shapefile covering the region under investigation.
`mesh.args`	list of parameters that control the mesh structure. *convex*, *concave* and *resolution*, to control the boundary of the inner mesh, and *max.edge*, *cut* and *offset*, to control the mesh itself, with the parameters having the same meaning as in the INLA functions *inla.convex.hull* and *inla.mesh.2d*.

Details

The mesh is created by finding a tight boundary around the polygon data, and creating a fine mesh within the boundary and a coarser mesh outside. This speeds up computation time by only having a very fine mesh within the area of interest and having a small region outside with a coarser mesh to avoid edge effects.

Six mesh parameters can be specified as arguments: *convex*, *concave* and *resolution*, to control the boundary of the inner mesh, and *max.edge*, *cut* and *offset*, to control the mesh itself, with the names meaning the same as used by INLA functions *inla.convex.hull* and *inla.mesh.2d*.

Defaults are: pars <- list(convex = -0.01, concave = -0.5, resolution = 300, max.edge = c(3.0, 8), cut = 0.4, offset = c(1, 15)).

Value

An inla.mesh object

Examples

```
## Not run:
 polygons <- list()
 for(i in 1:100) {
   row <- ceiling(i/10)
   col <- ifelse(i %% 10 != 0, i %% 10, 10)
   xmin = 2*(col - 1); xmax = 2*col; ymin = 2*(row - 1); ymax = 2*row
   polygons[[i]] <- rbind(c(xmin, ymax), c(xmax,ymax), c(xmax, ymin), c(xmin,ymin))
 }
```

```
 polys <- do.call(raster::spPolygons, polygons)
 response_df <- data.frame(area_id = 1:100, response = runif(100, min = 0, max = 10))
 spdf <- sp::SpatialPolygonsDataFrame(polys, response_df)

 my_mesh <- build_mesh(spdf)

## End(Not run)
```

`disaggregation-deprecated` *Deprecated functions in disaggregation*

Description

These functions still work but will be removed (defunct) in the next version.

Details

- `fit_model`: This function is deprecated, and will be removed in the next version of this package.

`dummy` *Roxygen commands*

Description

Roxygen commands

Usage

```
dummy()
```

`fit_model` *Fit the disaggregation model*

Description

fit_model function takes a *disag_data* object created by `prepare_data` and performs a Bayesian disaggregation fit.

Usage

```
fit_model(
  data,
  priors = NULL,
  family = "gaussian",
  link = "identity",
  iterations = 100,
  field = TRUE,
  iid = TRUE,
  hess_control_parscale = NULL,
  hess_control_ndeps = 1e-04,
  silent = TRUE
)

disag_model(
  data,
  priors = NULL,
  family = "gaussian",
  link = "identity",
  iterations = 100,
  field = TRUE,
  iid = TRUE,
  hess_control_parscale = NULL,
  hess_control_ndeps = 1e-04,
  silent = TRUE
)
```

Arguments

`data`	disag_data object returned by `prepare_data` function that contains all the necessary objects for the model fitting
`priors`	list of prior values
`family`	likelihood function: *gaussian*, *binomial* or *poisson*
`link`	link function: *logit*, *log* or *identity*
`iterations`	number of iterations to run the optimisation for
`field`	logical. Flag the spatial field on or off

`iid` logical. Flag the iid effect on or off

`hess_control_parscale`
Argument to scale parameters during the calculation of the Hessian. Must be the same length as the number of parameters. See `optimHess` for details.

`hess_control_ndeps`
Argument to control step sizes during the calculation of the Hessian. Either length 1 (same step size applied to all parameters) or the same length as the number of parameters. Default is 1e-3, try setting a smaller value if you get NaNs in the standard error of the parameters. See `optimHess` for details.

`silent` logical. Suppress verbose output.

Details

The model definition

The disaggregation model makes predictions at the pixel level:

$$link(pred_i) = \beta_0 + \beta X + GP(s_i) + u_i$$

And then aggregates these predictions to the polygon level using the weighted sum (via the aggregation raster, agg_i):

$$cases_j = \sum_{i \epsilon j} pred_i \times agg_i$$

$$rate_j = \frac{\sum_{i \epsilon j} pred_i \times agg_i}{\sum_{i \epsilon j} agg_i}$$

The different likelihood correspond to slightly different models (y_j is the response count data):

- Gaussian: If σ is the dispersion of the pixel data, σ_j is the dispersion of the polygon data, where $\sigma_j = \sigma \sqrt{\sum agg_i^2} / \sum agg_i$

  $$dnorm(y_j / \sum agg_i, rate_j, \sigma_j)$$

  - predicts incidence rate.
- Binomial: For a survey in polygon j, y_j is the number positive and N_j is the number tested.

  $$dbinom(y_j, N_j, rate_j)$$

  - predicts prevalence rate.
- Poisson:

  $$dpois(y_j, cases_j)$$

  - predicts incidence count.

Specify priors for the regression parameters, field and iid effect as a single list. Hyperpriors for the field are given as penalised complexity priors you specify ρ_{min} and ρ_{prob} for the range of the field where $P(\rho < \rho_{min}) = \rho_{prob}$, and σ_{min} and σ_{prob} for the variation of the field where $P(\sigma > \sigma_{min}) = \sigma_{prob}$. Also, specify pc priors for the iid effect

The *family* and *link* arguments are used to specify the likelihood and link function respectively. The likelihood function can be one of *gaussian*, *poisson* or *binomial*. The link function can be one of *logit*, *log* or *identity*. These are specified as strings.

The field and iid effect can be turned on or off via the *field* and *iid* logical flags. Both are default TRUE.

The *iterations* argument specifies the maximum number of iterations the model can run for to find an optimal point.

The *silent* argument can be used to publish/suppress verbose output. Default TRUE.

Value

A list is returned of class `disag_model`. The functions *summary*, *print* and *plot* can be used on `disag_model`. The list of class `disag_model` contains:

`obj`	The TMB model object returned by `MakeADFun`.
`opt`	The optimized model object returned by `nlminb`.
`sd_out`	The TMB object returned by `sdreport`.
`data`	The *disag_data* object used as an input to the model.
`model_setup`	A list of information on the model setup. Likelihood function (*family*), link function(*link*), logical: whether a field was used (*field*) and logical: whether an iid effect was used (*iid*).

References

Nanda et al. (2023) disaggregation: An R Package for Bayesian Spatial Disaggregation Modeling. <doi:10.18637/jss.v106.i11>

Examples

```
## Not run:
 polygons <- list()
 for(i in 1:100) {
  row <- ceiling(i/10)
  col <- ifelse(i %% 10 != 0, i %% 10, 10)
  xmin = 2*(col - 1); xmax = 2*col; ymin = 2*(row - 1); ymax = 2*row
  polygons[[i]] <- rbind(c(xmin, ymax), c(xmax,ymax), c(xmax, ymin), c(xmin,ymin))
 }

 polys <- do.call(raster::spPolygons, polygons)
 response_df <- data.frame(area_id = 1:100, response = runif(100, min = 0, max = 10))
 spdf <- sp::SpatialPolygonsDataFrame(polys, response_df)

 r <- raster::raster(ncol=20, nrow=20)
 r <- raster::setExtent(r, raster::extent(spdf))
r[] <- sapply(1:raster::ncell(r), function(x) rnorm(1, ifelse(x %% 20 != 0, x %% 20, 20), 3))
 r2 <- raster::raster(ncol=20, nrow=20)
 r2 <- raster::setExtent(r2, raster::extent(spdf))
 r2[] <- sapply(1:raster::ncell(r), function(x) rnorm(1, ceiling(x/10), 3))
 cov_rasters <- raster::stack(r, r2)
```

```
 cl <- parallel::makeCluster(2)
 doParallel::registerDoParallel(cl)
 test_data <- prepare_data(polygon_shapefile = spdf,
                           covariate_rasters = cov_rasters)
 parallel::stopCluster(cl)
 foreach::registerDoSEQ()

 result <- fit_model(test_data, iterations = 2)

## End(Not run)
```

`getCovariateRasters` *Get a RasterStack of covariates from a folder containing .tif files*

Description

Looks in a specified folder for raster files. Returns a RasterStack of the rasters cropped to the extent specified by the shape parameter.

Usage

```
getCovariateRasters(directory, file_pattern = ".tif$", shape)
```

Arguments

`directory`	Filepath to the directory containing the rasters.
`file_pattern`	Pattern the filenames must match. Default is all files ending in .tif .
`shape`	An object with an extent that the rasters will be cropped to.

Value

A RasterStack of the raster files in the directory

Examples

```
## Not run:
  getCovariateRasters('/home/rasters', '.tif$', shape)

## End(Not run)
```

`getPolygonData` *Extract polygon id and response data into a data.frame from a SpatialPolygonsDataFrame*

Description

Returns a data.frame with a row for each polygon in the SpatialPolygonDataFrame and columns: area_id, response and N, containing the id of the polygon, the values of the response for that polygon, and the sample size respectively. If the data is not survey data (the sample size does not exist), this column will contain NAs.

Usage

```
getPolygonData(
  shape,
  id_var = "area_id",
  response_var = "response",
  sample_size_var = NULL
)
```

Arguments

`shape`	A SpatialPolygons object containing response data.
`id_var`	Name of column in shape object with the polygon id. Default 'area_id'.
`response_var`	Name of column in shape object with the response data. Default 'response'.
`sample_size_var`	For survey data, name of column in SpatialPolygonDataFrame object (if it exists) with the sample size data. Default NULL.

Value

A data.frame with a row for each polygon in the SpatialPolygonDataFrame and columns: area_id, response and N, containing the id of the polygon, the values of the response for that polygon, and the sample size respectively. If the data is not survey data (the sample size does not exist), this column will contain NAs.

Examples

```
{
 polygons <- list()
 for(i in 1:100) {
   row <- ceiling(i/10)
   col <- ifelse(i %% 10 != 0, i %% 10, 10)
   xmin = 2*(col - 1); xmax = 2*col; ymin = 2*(row - 1); ymax = 2*row
   polygons[[i]] <- rbind(c(xmin, ymax), c(xmax,ymax), c(xmax, ymin), c(xmin,ymin))
 }
```

```
 polys <- do.call(raster::spPolygons, polygons)
 response_df <- data.frame(area_id = 1:100, response = runif(100, min = 0, max = 10))
 spdf <- sp::SpatialPolygonsDataFrame(polys, response_df)

 getPolygonData(spdf, id_var = 'area_id', response_var = 'response')
}
```

`getStartendindex` *Function to match pixels to their corresponding polygon*

Description

From the covariate data and polygon data, the function matches the polygon id between the two to find which pixels from the covariate data are contained in each of the polygons.

Usage

```
getStartendindex(covariates, polygon_data, id_var = "area_id")
```

Arguments

`covariates`	data.frame with each covariate as a column an and id column.
`polygon_data`	data.frame with polygon id and response data.
`id_var`	string with the name of the column in the covariate data.frame containing the polygon id.

Details

Takes a data.frame containing the covariate data with a polygon id column and one column for each covariate, and another data.frame containing polygon data with a polygon id, response and sample size column (as returned by `getPolygonData` function).

Returns a matrix with two columns and one row for each polygon. The first column is the index of the first row in covariate data that corresponds to that polygon, the second column is the index of the last row in covariate data that corresponds to that polygon.

Value

A matrix with two columns and one row for each polygon. The first column is the index of the first row in covariate data that corresponds to that polygon, the second column is the index of the last row in covariate data that corresponds to that polygon.

Examples

```
{
 covs <- data.frame(area_id = c(1, 1, 1, 2, 2, 3, 3, 3, 3), response = c(3, 9, 5, 2, 3, 6, 7, 3, 5))
 response <- data.frame(area_id = c(1, 2, 3), response = c(4, 7, 2), N = c(NA, NA, NA))
 getStartendindex(covs, response, 'area_id')
}
```

---

make_model_object — *Create the TMB model object for the disaggregation model*

Description

make_model_object function takes a *disag_data* object created by `prepare_data` and creates a TMB model object to be used in fitting.

Usage

```
make_model_object(
  data,
  priors = NULL,
  family = "gaussian",
  link = "identity",
  field = TRUE,
  iid = TRUE,
  silent = TRUE
)
```

Arguments

`data`	disag_data object returned by `prepare_data` function that contains all the necessary objects for the model fitting
`priors`	list of prior values
`family`	likelihood function: *gaussian*, *binomial* or *poisson*
`link`	link function: *logit*, *log* or *identity*
`field`	logical. Flag the spatial field on or off
`iid`	logical. Flag the iid effect on or off
`silent`	logical. Suppress verbose output.

Details

The model definition

The disaggregation model make predictions at the pixel level:

$$link(pred_i) = \beta_0 + \beta X + GP(s_i) + u_i$$

And then aggregates these predictions to the polygon level using the weighted sum (via the aggregation raster, agg_i):

$$cases_j = \sum_{i \epsilon j} pred_i \times agg_i$$

$$rate_j = \frac{\sum_{i \epsilon j} pred_i \times agg_i}{\sum_{i \epsilon j} agg_i}$$

The different likelihood correspond to slightly different models (y_j is the response count data):

- Gaussian: If σ is the dispersion of the pixel data, σ_j is the dispersion of the polygon data, where $\sigma_j = \sigma \sqrt{\sum agg_i^2} / \sum agg_i$

  $$dnorm(y_j / \sum agg_i, rate_j, \sigma_j)$$

  - predicts incidence rate.
- Binomial: For a survey in polygon j, y_j is the number positive and N_j is the number tested.

  $$dbinom(y_j, N_j, rate_j)$$

  - predicts prevalence rate.
- Poisson:

  $$dpois(y_j, cases_j)$$

  - predicts incidence count.

Specify priors for the regression parameters, field and iid effect as a single named list. Hyperpriors for the field are given as penalised complexity priors you specify ρ_{min} and ρ_{prob} for the range of the field where $P(\rho < \rho_{min}) = \rho_{prob}$, and σ_{min} and σ_{prob} for the variation of the field where $P(\sigma > \sigma_{min}) = \sigma_{prob}$. Also, specify pc priors for the iid effect.

The precise names and default values for these priors are:

- priormean_intercept: 0
- priorsd_intercept: 10.0
- priormean_slope: 0.0
- priorsd_slope: 0.5
- prior_rho_min: A third the length of the diagonal of the bounding box.
- prior_rho_prob: 0.1
- prior_sigma_max: sd(response/mean(response))
- prior_sigma_prob: 0.1
- prior_iideffect_sd_max: 0.1

- prior_iideffect_sd_prob: 0.01

The *family* and *link* arguments are used to specify the likelihood and link function respectively. The likelihood function can be one of *gaussian*, *poisson* or *binomial*. The link function can be one of *logit*, *log* or *identity*. These are specified as strings.

The field and iid effect can be turned on or off via the *field* and *iid* logical flags. Both are default TRUE.

The *iterations* argument specifies the maximum number of iterations the model can run for to find an optimal point.

The *silent* argument can be used to publish/supress verbose output. Default TRUE.

Value

The TMB model object returned by MakeADFun.

Examples

```
## Not run:
 polygons <- list()
 for(i in 1:100) {
  row <- ceiling(i/10)
  col <- ifelse(i %% 10 != 0, i %% 10, 10)
  xmin = 2*(col - 1); xmax = 2*col; ymin = 2*(row - 1); ymax = 2*row
  polygons[[i]] <- rbind(c(xmin, ymax), c(xmax,ymax), c(xmax, ymin), c(xmin,ymin))
 }

 polys <- do.call(raster::spPolygons, polygons)
 response_df <- data.frame(area_id = 1:100, response = runif(100, min = 0, max = 10))
 spdf <- sp::SpatialPolygonsDataFrame(polys, response_df)

 r <- raster::raster(ncol=20, nrow=20)
 r <- raster::setExtent(r, raster::extent(spdf))
r[] <- sapply(1:raster::ncell(r), function(x) rnorm(1, ifelse(x %% 20 != 0, x %% 20, 20), 3))
 r2 <- raster::raster(ncol=20, nrow=20)
 r2 <- raster::setExtent(r2, raster::extent(spdf))
 r2[] <- sapply(1:raster::ncell(r), function(x) rnorm(1, ceiling(x/10), 3))
 cov_rasters <- raster::stack(r, r2)

 cl <- parallel::makeCluster(2)
 doParallel::registerDoParallel(cl)
 test_data <- prepare_data(polygon_shapefile = spdf,
                           covariate_rasters = cov_rasters)
 parallel::stopCluster(cl)
 foreach::registerDoSEQ()

 result <- make_model_object(test_data)

## End(Not run)
```

`parallelExtract` *Parallel extraction of raster stack by shape file.*

Description

Parallelisation is performed across rasters, not shapes. So this function is only useful if you are extracting data from many raster layers. As the overhead for parallel computation in windows is high it only makes sense to parallelise in this way.

Usage

```
parallelExtract(raster, shape, fun = mean, id = "OBJECTID", ...)
```

Arguments

`raster`	A RasterBrick or RasterStack object.
`shape`	A SpatialPolygons object.
`fun`	The function used to aggregate the pixel data. If NULL, raw pixel data is returned.
`id`	Name of column in shape object to be used to bind an ID column to output.
`...`	Other arguments to raster::extract.

Value

A data.frame with columns of polygon id, cell id (if fun = NULL) and a column for each raster in the stack

Examples

```
## Not run:
 polygons <- list()
 for(i in 1:100) {
   row <- ceiling(i/10)
   col <- ifelse(i %% 10 != 0, i %% 10, 10)
   xmin = 2*(col - 1); xmax = 2*col; ymin = 2*(row - 1); ymax = 2*row
   polygons[[i]] <- rbind(c(xmin, ymax), c(xmax,ymax), c(xmax, ymin), c(xmin,ymin))
 }

 polys <- do.call(raster::spPolygons, polygons)
 response_df <- data.frame(area_id = 1:100, response = runif(100, min = 0, max = 10))
 spdf <- sp::SpatialPolygonsDataFrame(polys, response_df)

 r <- raster::raster(ncol=20, nrow=20)
 r <- raster::setExtent(r, raster::extent(spdf))
r[] <- sapply(1:raster::ncell(r), function(x) rnorm(1, ifelse(x %% 20 != 0, x %% 20, 20), 3))
 r2 <- raster::raster(ncol=20, nrow=20)
 r2 <- raster::setExtent(r2, raster::extent(spdf))
 r2[] <- sapply(1:raster::ncell(r), function(x) rnorm(1, ceiling(x/10), 3))
```

```
  cov_rasters <- raster::stack(r, r2)

  cl <- parallel::makeCluster(2)
  doParallel::registerDoParallel(cl)
  result <- parallelExtract(cov_rasters, spdf, fun = NULL, id = 'area_id')
  parallel::stopCluster(cl)
  foreach::registerDoSEQ()

## End(Not run)
```

`plot.disag_data` *Plot input data for disaggregation*

Description

Plotting function for class *disag_data* (the input data for disaggregation).

Usage

```
## S3 method for class 'disag_data'
plot(x, which = c(1, 2, 3), ...)
```

Arguments

`x`	Object of class *disag_data* to be plotted.
`which`	If a subset of plots is required, specify a subset of the numbers 1:3
`...`	Further arguments to *plot* function.

Details

Produces three plots: polygon response data, covariate rasters and INLA mesh.

Value

A list of three plots: the polygon plot (ggplot), covariate plot (spplot) and INLA mesh plot (ggplot)

`plot.disag_model` *Plot results of fitted model*

Description

Plotting function for class *disag_model* (the result of the disaggregation fitting).

Usage

```
## S3 method for class 'disag_model'
plot(x, ...)
```

Arguments

`x`	Object of class *disag_model* to be plotted.
`...`	Further arguments to *plot* function.

Details

Produces two plots: results of the fixed effects and in-sample observed vs predicted plot.

Value

A list of two ggplot plots: results of the fixed effects and an in-sample observed vs predicted plot

`plot.disag_prediction` *Plot mean and uncertainty predictions from the disaggregation model results*

Description

Plotting function for class *disag_prediction* (the mean and uncertainty predictions of the disaggregation fitting).

Usage

```
## S3 method for class 'disag_prediction'
plot(x, ...)
```

Arguments

`x`	Object of class *disag_prediction* to be plotted.
`...`	Further arguments to *plot* function.

Details

Produces raster plots of the mean prediction, and the lower and upper confidence intervals.

Value

A list of plots of rasters from the prediction: mean prediction, lower CI and upper CI.

`predict.disag_model` *Predict mean and uncertainty from the disaggregation model result*

Description

predict.disag_model function takes a *disag_model* object created by *disaggregation::disag_model* and predicts mean and uncertainty maps.

Usage

```
## S3 method for class 'disag_model'
predict(object, newdata = NULL, predict_iid = FALSE, N = 100, CI = 0.95, ...)
```

Arguments

`object`	disag_model object returned by disag_model function.
`newdata`	If NULL, predictions are made using the data in model_output. If this is a raster stack or brick, predictions will be made over this data.
`predict_iid`	logical. If TRUE, any polygon iid effect from the model will be used in the prediction. Default FALSE.
`N`	Number of realisations. Default: 100.
`CI`	Confidence interval to be calculated from the realisations. Default: 0.95.
`...`	Further arguments passed to or from other methods.

Details

To predict over a different spatial extent to that used in the model, a RasterStack covering the region to make predictions over is passed to the argument *newdata*. If this is not given predictions are made over the data used in the fit.

The *predict_iid* logical flag should be set to TRUE if the results of the iid effect from the model are to be used in the prediction.

For the uncertainty calculations, the number of the realisations and the size of the confidence interval to be calculated are given by the arguments *N* and *CI* respectively.

Value

An object of class *disag_prediction* which consists of a list of two objects:

`mean_prediction`
: List of:

- *prediction* Raster of mean predictions based.
- *field* Raster of the field component of the linear predictor.
- *iid* Raster of the iid component of the linear predictor.
- *covariates* Raster of the covariate component of the linear predictor.

`uncertainty_prediction:`
: List of:

- *realisations* RasterStack of realisations of predictions. Number of realisations defined by argument *N*.
- *predictions_ci* RasterStack of the upper and lower credible intervals. Defined by argument *CI*.

Examples

```
## Not run:
predict(fit_result)

## End(Not run)
```

`predict_model` *Function to predict mean from the model result*

Description

predict_model function takes a *disag_model* object created by *disaggregation::disag_model* and predicts mean maps.

Usage

```
predict_model(model_output, newdata = NULL, predict_iid = FALSE)
```

Arguments

`model_output`	disag_model object returned by disag_model function
`newdata`	If NULL, predictions are made using the data in model_output. If this is a raster stack or brick, predictions will be made over this data. Default NULL.
`predict_iid`	If TRUE, any polygon iid effect from the model will be used in the prediction. Default FALSE.

Details

Function returns rasters of the mean predictions as well as the covariate and field contributions to the linear predictor.

To predict over a different spatial extent to that used in the model, a RasterStack covering the region to make predictions over is passed to the argument *newdata*. If this is not given predictions are made over the data used in the fit.

The *predict_iid* logical flag should be set to TRUE if the results of the iid effect from the model are to be used in the prediction.

Value

The mean prediction, which is a list of:

- *prediction* Raster of mean predictions based.
- *field* Raster of the field component of the linear predictor.
- *iid* Raster of the iid component of the linear predictor.
- *covariates* Raster of the covariate component of the linear predictor.

Examples

```
## Not run:
predict_model(result)

## End(Not run)
```

---

predict_uncertainty *Function to predict uncertainty from the model result*

Description

predict_uncertainty function takes a *disag_model* object created by *disaggregation::disag_model* and predicts upper and lower credible interval maps.

Usage

```
predict_uncertainty(
  model_output,
  newdata = NULL,
  predict_iid = FALSE,
  N = 100,
  CI = 0.95
)
```

Arguments

`model_output`	disag_model object returned by disag_model function.
`newdata`	If NULL, predictions are made using the data in model_output. If this is a raster stack or brick, predictions will be made over this data. Default NULL.
`predict_iid`	If TRUE, any polygon iid effect from the model will be used in the prediction. Default FALSE.
`N`	number of realisations. Default: 100.
`CI`	confidence interval. Default: 0.95.

Details

Function returns a RasterStack of the realisations as well as the upper and lower credible interval rasters.

To predict over a different spatial extent to that used in the model, a RasterStack covering the region to make predictions over is passed to the argument *newdata*. If this is not given predictions are made over the data used in the fit.

The *predict_iid* logical flag should be set to TRUE if the results of the iid effect from the model are to be used in the prediction.

The number of the realisations and the size of the confidence interval to be calculated. are given by the arguments *N* and *CI* respectively.

Value

The uncertainty prediction, which is a list of:

- *realisations* RasterStack of realisations of predictions. Number of realisations defined by argument *N*.
- *predictions_ci* RasterStack of the upper and lower credible intervals. Defined by argument *CI*.

Examples

```
## Not run:
predict_uncertainty(result)

## End(Not run)
```

`prepare_data` *Prepare data for disaggregation modelling*

Description

prepare_data function is used to extract all the data required for fitting a disaggregation model. Designed to be used in the *disaggregation::fit_model* function.

Usage

```
prepare_data(
  polygon_shapefile,
  covariate_rasters,
  aggregation_raster = NULL,
  id_var = "area_id",
  response_var = "response",
  sample_size_var = NULL,
  mesh.args = NULL,
  na.action = FALSE,
  makeMesh = TRUE,
  ncores = 2
)
```

Arguments

`polygon_shapefile`
: SpatialPolygonDataFrame containing at least two columns: one with the id for the polygons (*id_var*) and one with the response count data (*response_var*); for binomial data, i.e survey data, it can also contain a sample size column (*sample_size_var*).

`covariate_rasters`
: RasterStack of covariate rasters to be used in the model.

`aggregation_raster`
: Raster to aggregate pixel level predictions to polygon level e.g. population to aggregate prevalence. If this is not supplied a uniform raster will be used.

`id_var`
: Name of column in SpatialPolygonDataFrame object with the polygon id.

`response_var`
: Name of column in SpatialPolygonDataFrame object with the response data.

`sample_size_var`
: For survey data, name of column in SpatialPolygonDataFrame object (if it exists) with the sample size data.

`mesh.args`
: list of parameters that control the mesh structure with the same names as used by INLA.

`na.action`
: logical. If TRUE, NAs in response will be removed, covariate NAs will be given the median value, aggregation NAs will be set to zero. Default FALSE (NAs in response or covariate data within the polygons will give errors).

`makeMesh`
: logical. If TRUE, build INLA mesh, takes some time. Default TRUE.

`ncores`
: Number of cores used to perform covariate extraction.

Details

Takes a SpatialPolygonDataFrame with the response data and a RasterStack of covariates.

Extract the values of the covariates (as well as the aggregation raster, if given) at each pixel within the polygons (*parallelExtract* function). This is done in parallel and *n.cores* argument is used to set the number of cores to use for covariate extraction. This can be the number of covariates used in the model.

The aggregation raster defines how the pixels within each polygon are aggregated. The disaggregation model performs a weighted sum of the pixel prediction, weighted by the pixel values in the aggregation raster. For disease incidence rate you use the population raster to aggregate pixel incidence rate by summing the number of cases (rate weighted by population). If no aggregation raster is provided a uniform distribution is assumed, i.e. the pixel predictions are aggregated to polygon level by summing the pixel values.

Makes a matrix that contains the start and end pixel index for each polygon. Builds an INLA mesh to use for the spatial field (*getStartendindex* function).

The *mesh.args* argument allows you to supply a list of INLA mesh parameters to control the mesh used for the spatial field (*build_mesh* function).

The *na.action* flag is automatically off. If there are any NAs in the response or covariate data within the polygons the *prepare_data* method will error. Ideally the NAs in the data would be dealt with beforehand, however, setting na.action = TRUE will automatically deal with NAs. It removes any polygons that have NAs as a response, sets any aggregation pixels with NA to zero and sets covariate NAs pixels to the median value for the that covariate.

Value

A list is returned of class `disag_data`. The functions *summary*, *print* and *plot* can be used on `disag_data`. The list of class `disag_data` contains:

`polygon_shapefile`
: The SpatialPolygonDataFrame used as an input.

`covariate_rasters`
: The RasterStack used as an input.

`polygon_data`
: A data frame with columns of *area_id*, *response* and *N* (sample size: all NAs unless using binomial data). Each row represents a polygon.

`covariate_data`
: A data frame with columns of *area_id*, *cell_id* and one for each covariate in *covariate_rasters*. Each row represents a pixel in a polygon.

`aggregation_pixels`
: An array with the value of the aggregation raster for each pixel in the same order as the rows of *covariate_data*.

`coordsForFit`
: A matrix with two columns of x, y coordinates of pixels within the polygons. Used to make the spatial field.

`coordsForPrediction`
: A matrix with two columns of x, y coordinates of pixels in the whole Raster. Used to make predictions.

`startendindex`
: A matrix with two columns containing the start and end index of the pixels within each polygon.

`mesh`
: A INLA mesh to be used for the spatial field of the disaggregation model.

Examples

```
polygons <- list()
for(i in 1:100) {
```

```
  row <- ceiling(i/10)
  col <- ifelse(i %% 10 != 0, i %% 10, 10)
  xmin = 2*(col - 1); xmax = 2*col; ymin = 2*(row - 1); ymax = 2*row
  polygons[[i]] <- rbind(c(xmin, ymax), c(xmax,ymax), c(xmax, ymin), c(xmin,ymin))
 }

 polys <- do.call(raster::spPolygons, polygons)
 response_df <- data.frame(area_id = 1:100, response = runif(100, min = 0, max = 10))
 spdf <- sp::SpatialPolygonsDataFrame(polys, response_df)

 r <- raster::raster(ncol=20, nrow=20)
 r <- raster::setExtent(r, raster::extent(spdf))
 r[] <- sapply(1:raster::ncell(r), function(x) rnorm(1, ifelse(x %% 20 != 0, x %% 20, 20), 3))
 r2 <- raster::raster(ncol=20, nrow=20)
 r2 <- raster::setExtent(r2, raster::extent(spdf))
 r2[] <- sapply(1:raster::ncell(r), function(x) rnorm(1, ceiling(x/10), 3))
 cov_rasters <- raster::stack(r, r2)

 test_data <- prepare_data(polygon_shapefile = spdf,
                           covariate_rasters = cov_rasters)
```

print.disag_data — *Print function for disaggregation input data*

Description

Function that prints the input data from the disaggregation model.

Usage

```
## S3 method for class 'disag_data'
print(x, ...)
```

Arguments

x	Object returned from prepare_data.
...	Further arguments to *print* function.

Details

Prints the number of polyons and pixels, the number of pixels in the largest and smallest polygons and summaries of the covariates.

`print.disag_model` *Print function for disaggregation fit result.*

Description

Function that prints the result of the fit from the disaggregation model.

Usage

```
## S3 method for class 'disag_model'
print(x, ...)
```

Arguments

`x`	Object returned from disag_model.
`...`	Further arguments to *print* function.

Details

Prints the negative log likelihood, model parameters and calculates metrics from in-sample performance.

`print.disag_prediction` *Print function for disaggregation prediction*

Description

Function that prints the prediction from the disaggregation model.

Usage

```
## S3 method for class 'disag_prediction'
print(x, ...)
```

Arguments

`x`	Object returned from predict.disag_model.
`...`	Further arguments to *print* function.

Details

Prints the number of polyons and pixels, the number of pixels in the largest and smallest polygons and summaries of the covariates.

summary.disag_data *Summary function for disaggregation input data*

Description

Function that summarizes the input data from the disaggregation model.

Usage

```
## S3 method for class 'disag_data'
summary(object, ...)
```

Arguments

`object`	Object returned from prepare_data.
`...`	Further arguments to *summary* function.

Details

Prints the number of polyons and pixels, the number of pixels in the largest and smallest polygons and summaries of the covariates.

Value

A list of the number of polyons, the number of covariates and summaries of the covariates.

summary.disag_model *Summary function for disaggregation fit result*

Description

Function that summarises the result of the fit from the disaggregation model.

Usage

```
## S3 method for class 'disag_model'
summary(object, ...)
```

Arguments

`object`	Object returned from disag_model.
`...`	Further arguments to *summary* function.

Details

Prints the negative log likelihood, model parameters and calculates metrics from in-sample performance.

Value

A list of the model parameters, negative log likelihood and metrics from in-sample performance.

---

summary.disag_prediction

Summary function for disaggregation prediction

Description

Function that summarizes the prediction from the disaggregation model.

Usage

```
## S3 method for class 'disag_prediction'
summary(object, ...)
```

Arguments

`object`	Object returned from predict.disag_model
`...`	Further arguments to *summary* function.

Details

Prints the number of polyons and pixels, the number of pixels in the largest and smallest polygons and summaries of the covariates.

Value

A list of the number of polyons, the number of covariates and summaries of the covariates.

Index

as.disag_data, 2

build_mesh, 4

disag_model *(*fit_model*)*, 6
disaggregation-deprecated, 5
dummy, 5

fit_model, *5*, 6

getCovariateRasters, 9
getPolygonData, 10
getStartendindex, 11

make_model_object, 12
MakeADFun, *8*, *14*

nlminb, *8*

optimHess, *7*

parallelExtract, 15
plot.disag_data, 16
plot.disag_model, 17
plot.disag_prediction, 17
predict.disag_model, 18
predict_model, 19
predict_uncertainty, 20
prepare_data, *6*, *12*, 21
print.disag_data, 24
print.disag_model, 25
print.disag_prediction, 25

sdreport, *8*
summary.disag_data, 26
summary.disag_model, 26
summary.disag_prediction, 27