Package 'zoon'

February 28, 2020

Type Package

Title Reproducible, Accessible & Shareable Species Distribution Modelling

Version 0.6.5

Date 2020-02-27

Maintainer Tom August <tomaug@ceh.ac.uk>

Description Reproducible and remixable species distribution modelling. The package reads user submitted modules from an online repository, runs full species distribution modelling workflows and returns output that is fully reproducible. For examples and detailed discussion refer to: N.Golding et al. (2017) 'The zoon r package for reproducible and shareable species distribution modelling'. Methods in Ecology and Evolution. <doi:10.1111/2041-210X.12858>. The package 'SDMTools' is used for testing and, though this package is archived, you can access it here if needed, <https://cran.r-project.org/src/contrib/Archive/SDMTools/>.

License BSD_3_clause + file LICENSE

Imports dismo, methods, plyr, randomForest, RCurl, rfigshare, rgdal, roxygen2, rworldmap, sp, testthat

Depends raster (>= 2.4-20), R (>= 3.2.0)

Suggests knitr, maxlike, SDMTools

VignetteBuilder knitr

LazyData TRUE

URL https://github.com/zoonproject/zoon

BugReports https://github.com/zoonproject/zoon/issues

RoxygenNote 7.0.2

NeedsCompilation no

Author Tom August [aut, cre],
Tim Lucas [aut] (<https://orcid.org/0000-0003-4694-8107>),
Nick Golding [aut],
Emiel van Loon [ctb],
Greg McInerny [aut]

Repository CRAN

Date/Publication 2020-02-28 16:30:02 UTC

R topics documented:

`AplumbeusOcc` *UK occurrence data for Anopheles plumbeus as taken from GBIF*

Description

This is an example occurrence only data for the species Anopheles plumbeus. The data are taken from GBIF and restricted to the UK. These data are used in the module UKAnophelesPlumbeus which makes for a quick running occurrence module for testing and playing with zoon.

Format

data.frame with five columns, longitude, latitude, value (1 for presence), type (presence) and a column of 1s indicating this is training data not external validation data.

Author(s)

Tim Lucas September 2014

Source

GBIF

`BuildModule` *BuildModule*

Description

Turn a function in the namespace into a module. Will later add functions to upload module to figshare etc. And add testing that the module name is unique.

Usage

```
BuildModule(
  object,
  type,
  dir = ".",
  title = "",
  description = "",
  details = "",
  author = "",
  email = "",
  version = 0.1,
  paras = NULL,
  dataType = NULL,
  check = TRUE
)
```

Arguments

`object`	A function that will be made into a module file. It is good practice to ensure your function does not have the same name as a base function, another module, or other common functions.
`type`	A string that defines the type of module. Possible module types are occurrence, covariate, process, model, and output.
`dir`	The directory to put the module into (defaults to the working directory).
`title`	A short description of the module.
`description`	(required) A single string giving a full description of the module.
`details`	(optional) A single string giving details of the module.
`author`	(required) String giving the author(s) name(s)
`email`	(required) String giving the correspondence address for the module (only give one address).
`version`	(optional) Numeric giving the version number. Default 0.1.
`paras`	A list of the form list(parameterName = 'Parameter description.', anotherParameter = 'Another description.') This is required if the module takes non-default arguments
`dataType`	Character vector required for all module types except 'covariate'. Indicates the types of data that this module works with. Values can be any of 'presence-only', 'presence/absence', 'presence/background', 'abundance' or 'proportion'. For a occurrence model this should indicate the type of data that is returned and for other modules should indicate the type of data they will work with. If the module works with multiple types they can be supplied in a vector, e.g. c('presence-only', 'presence/absence')
`check`	Logical indicating if the module should be run through checks once it has been built. Defaults to TRUE.

Value

Name of the module. As a side effect outputs a .R file to the directory specified.

`cbindZoon` *cbindZoon*

Description

cbindZoon

Usage

```
cbindZoon(a, b)
```

Arguments

`a`	The dataframe of which you'd like preserve the user defined attributes
`b`	The dataframe which you'd like to append to a.

`Chain` *Chain modules together*

Description

`Chain` combines multiple modules of the same module type such that they are executed sequentially and their outputs combined. For example, process modules may be `Chain`ed to carry out successive processing operations. By contrast, `list`ing modules of the same type would split the workflow into multiple parallel workflows, each using a different module at this step.

Usage

```
Chain(...)
```

Arguments

`...`	List of modules to be chained.

Details

Similarly for occurrence or covariate modules the datasets are joined (row- or layer-wise) whereas `list` would carry out separate analyses. Model and output modules may not be chained. Developers should note that this function is not actually used - calls using `Chain` are parsed by workflow, with behaviour similar to this function.

`ChangeWorkflow` *Change a workflow and rerun.*

Description

Takes a workflow object and reruns it with changes.

Usage

```
ChangeWorkflow(
  workflow,
  occurrence = NULL,
  covariate = NULL,
  process = NULL,
  model = NULL,
  output = NULL,
  forceReproducible = NULL
)
```

Arguments

`workflow`	A zoonWorkflow object from a previous zoon analysis
`occurrence, covariate, process, model, output`	Optional modules (or lists or Chains) to replace the modules in `workflow`
`forceReproducible`	Optional logical. Whether to force zoon to collect modules from the online repo in the new workflow. This ensure the analysis is reproducible.

Value

A list with the results of each module and a copy of the call used to execute the workflow (

Examples

```
## Not run:
w <- workflow(UKAnophelesPlumbeus,
              UKAir,
              Background(n = 70),
              LogisticRegression,
              PrintMap)

w2 <- ChangeWorkflow(w,
                     output = PrintMap)

## End(Not run)
```

`CombineRasters` *Combine rasters*

Description

A generalised function to combine rasters of differing CRS and resolution. The function takes and returns a list of Raster* objects. The final CRS is the one most frequent in the rasters being combined, or lat/long if there is no CRS is most frequent

Usage

```
CombineRasters(rasters, method = "ngb")
```

Arguments

`rasters`	A list of Raster* objects to be converted to a common CRS and resolution
`method`	The method used in raster::projectRaster. Either 'ngb' (nearest neighbor), which is useful for categorical variables, or 'bilinear' (bilinear interpolation; the default value), which is appropriate for continuous variables

Value

A list of rasters

CWBZim *Presence/absence of the coffee white stem borer in Zimbabwe 2003*

Description

This is an example presence/absence dataset for the coffee white stem borer *Monochamus leuconotus* P. taken from an open access dataset on the Dryad data repository. The data are made available by those authors under a Creative Commons CC0 These data are used in the module CWBZimbabwe which can be used for running toy presence/absence species distribution models.

Format

data.frame with five columns, longitude, latitude, value (1 for presence), type (presence) and a column of 1s indicating this is training data not external validation data.

Author(s)

Nick Golding August 2015

Source

Original publication: Kutywayo D, Chemura A, Kusena W, Chidoko P, Mahoya C (2013) The impact of climate change on the potential distribution of agricultural pests: the case of the coffee white stem borer (*Monochamus leuconotus* P.) in Zimbabwe. PLoS ONE 8(8): e73432. Dryad data package: Kutywayo D, Chemura A, Kusena W, Chidoko P, Mahoya C (2013) Data from: The impact of climate change on the potential distribution of agricultural pests: the case of the coffee white stem borer (*Monochamus leuconotus* P.) in Zimbabwe. Dryad Digital Repository. `http://dx.doi.org/10.1371/journal.pone.0073432`

ExtractAndCombData *ExtractAndCombData*

Description

Simply extract covariates from rasters and combine with occurrence data. This function is primarily used internally but can be used when running workflows interactively (see vignette `basic zoon useage`)

Usage

```
ExtractAndCombData(occurrence, ras)
```

Arguments

`occurrence`	A data frame from an occurrence module
`ras`	Environmental raster layer, stack or brick.

FindExtent *Determine the Extent of Interest*

Description

Helper function to define an extent (in latitude and longitude) describing the area of interest for modelling. Opens a world map, on which you can click to twice to determine the area of interest.

Usage

```
FindExtent(
  initial_extent = c(-180, 180, -90, 90),
  resolution = c("low", "medium"),
  round = 3
)
```

Arguments

`initial_extent`	optional numeric vector or extent object defining the giving the extent (in latitude and longitude) of the world map to plot. This should be larger than the target extent so that you can click within it. Can be useful for 'zooming in' to an area to define an extent more precisely.
`resolution`	how detailed to make national borders in the plotted world map. "low" is less accurate, but faster to load than "medium"
`round`	to how many decimal places the extent should be reported. The default value rounds to 3 decimal places. Set this to `Inf` to prevent any rounding. This only affects the vector version of the extent printed in the console, not the extent object returned.

Details

This is just a thin wrapper around `raster::getExtent`, providing the world map to click on and reporting the extent as a rounded vector.

Value

invisibly returns an extent object, also prints a vector version of that extent to the console.

Examples

```
## Not run:
 # open a world map and click on two spots to print the extent to the console
 FindExtent()

 # you can get the corresponding extent object too
 ext <- FindExtent()
 ext
 # you can zoom in on an area and increase the resolution, and precision of
 # the extent vector
 FindExtent(c(112, 156, -44, -8),
            resolution = "medium",
            round = 6)

## End(Not run)
```

`FrescaloBias` *Example bias raster for plants in England*

Description

This is an example bias raster giving a modelled estimate of the relative recording effort for plants in England using the Frescalo function in the R package sparta.

Format

RasterLayer with extent: c(-7.083, 2.167, 49.83, 55.83) and values ranging between 0 and 1.

Author(s)

Tom August & Nick Golding September 2015

`GetMaxEnt` *Get MaxEnt*

Description

Helper function to get the MaxEnt java executable and install it in the right locations for zoon modules that use ‘dismo::maxent‘ and ‘biomod2‘.

Usage

```
GetMaxEnt()
```

Details

Since MaxEnt may not be distributed other than via the MaxEnt website, users must download the file themselves and place it in the right location for R packages to access it. This function helps with that. Just run `GetMaxEnt()` and follow the prompts in the terminal.

GetModuleList *GetModuleList*

Description

Get a list of all the modules available on the github repo.

Usage

```
GetModuleList(
  type = c("all", "occurrence", "covariate", "process", "model", "output"),
  renew = FALSE
)
```

Arguments

`type`	The subset of zoon modules you want to return. Defaults to 'all', but you can select any of the zoon workflow steps: 'occurrence', 'covariate', 'process', 'model', or 'output'.
`renew`	Download from github even if we already have a module list.

Details

This function will only work on a platform that supports the method 'libcurl' in the function url. This can be tested using the function `capabilities` (see example).

Value

A list with all module names.

Examples

```
# GetModuleList requires libcurl to be supported
if(capabilities('libcurl')) GetModuleList()
```

GetPackage *Helper to install (if needed) and load a package*

Description

Given a package name, either as a string or object, load the package if it exists, else install it from CRAN and then load

Usage

```
GetPackage(package)
```

Arguments

`package`	A character vector of packages to load

Examples

```
## Not run:

GetPackage('gam')


## End(Not run)
```

`LoadModule` *A function to load a module function from url or disk.*

Description

Loads a module function into the global environment ready to be used in a zoon workflow. This function is mostly for use while developing modules. Workflows run with modules defined locally are no longer reproducible and so are discouraged and will be tagged as 'unreproducible'.

Usage

```
LoadModule(module)
```

Arguments

`module`	A string that describes the location of the R file. Can be a a full URL or a path to a local file.

Value

Name of the function. Adds function to global namespace.

`ModuleArguments` *ModuleArguments*

Description

Produce list of module arguments

Usage

```
ModuleArguments(ModuleName)
```

Arguments

`ModuleName`	string giving the name of the module

Value

A list of arguments for user-intput to that module

Examples

```
ModuleArguments('Background')
```

`ModuleHelp` *ModuleHelp*

Description

Returns the help file for a zoon module.

Usage

```
ModuleHelp(module)
```

Arguments

`module`	The name of a zoon module

Value

Prints the help page to screen.

See Also

`GetModuleList`

`Occurrence` *Accessor functions for getting module outputs from a workflow object*

Description

These functions access the output from each module type. If workflows are split using list, they will return a list with the output of each separate workflow being one element of the list.

Usage

```
Occurrence(workflow)

Covariate(workflow)

Process(workflow)

Model(workflow)

Output(workflow)
```

Arguments

`workflow`	A workflow object

Examples

```
## Not run:
work1 <- workflow(occurrence = UKAnophelesPlumbeus,
                  covariate  = UKAir,
                  process    = Background(n = 70),
                  model      = list(LogisticRegression, LogisticRegression),
                  output     = PrintMap)

Occurrence(work1)
Covariate(work1)
Process(work1)
Model(work1)
Model(work1)[[1]]
Output(work1)

## End(Not run)
```

`plot.zoonWorkflow` *Plot a schematic illustration of a zoon workflow structure.*

Description

Opens a graphics device and produces a plot of the workflow structure and modules used.

Usage

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

Arguments

`x`	an object of class zoonWorkflow
`...`	currently ignored

`print.zoonCitation` *A function to print zoonCitation*

Description

Prints a zoonCitation object to console giving easy access to module citations

Usage

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

Arguments

`x`	object of class zoonCitation
`...`	currently ignored

`print.zoonWorkflow` *A function to print a zoonWorkflow object*

Description

The function returns a very simple output detailing the function call.

Usage

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

Arguments

`x`	object of class zoonWorkflow
`...`	currently ignored

`Replicate` *Replicate a module multiple times*

Description

This function is useful when running simulations and could be used with modules that have random number generation internally meaning that results from identical runs are different. Replicate gives the same result as using list() and repeating the module multiple times.

Usage

```
Replicate(call, n)
```

Arguments

`call`	A module call, e.g. UKAnophelesPlumbeus
`n`	The number of times to replicate the call

Value

A list of calls

Examples

```
# run a workflow, using the logistic regression model
## Not run:

Without Replicate
work1 <- workflow(occurrence = list(UKAnophelesPlumbeus,UKAnophelesPlumbeus,UKAnophelesPlumbeus),
               covariate = UKAir,
               process = OneHundredBackground,
               model = LogisticRegression,
               output = SameTimePlaceMap)
# With Replicate
work2 <- workflow(occurrence = Replicate(UKAnophelesPlumbeus, 3),
               covariate = UKAir,
               process = OneHundredBackground,
               model = LogisticRegression,
               output = SameTimePlaceMap)
# The workflows are the same
plot(work1)
plot(work2)

# Output plots show the random placement of background points
# in each run
work1 <- workflow(occurrence = UKAnophelesPlumbeus,
                 covariate = UKAir,
                 process = Replicate(Background(n=10), n = 10),
                 model = LogisticRegression,
                 output = PrintMap)


## End(Not run)
```

`RerunWorkflow` *Rerun a workflow object.*

Description

Takes a workflow object and reruns it.

Usage

```
RerunWorkflow(workflow, from = NULL)
```

Arguments

`workflow`	A zoonWorkflow object from a previous zoon analysis
`from`	Which modules should be run. If NULL (default), run from the first NULL output (i.e. where the workflow broke). Otherwise takes an integer and runs from that module.

Value

A list with the results of each module and a copy of the call used to execute the workflow.

Examples

```
## Not run:
w <- workflow(UKAnophelesPlumbeus,
              UKAir,
              Background(n = 70),
              LogisticRegression,
              PrintMap)

RerunWorkflow(w)

## End(Not run)
```

`RunModels` *RunModels*

Description

A function to train and predict crossvalidation folds and train one full model and predict any external validation data. This function is primarily used internally but can be used when running workflows interactively (see vignette `basic zoon useage`)

Usage

```
RunModels(df, modelFunction, paras, workEnv)
```

Arguments

`df`	Dataframe from process module. Should contain columns value, type, lon, lat and fold, a number indicating which cross validation set a datapoint is in. If fold is 0 then this is considered external validation data If all data is 0 or 1, then no cross validation is run.
`modelFunction`	String giving the name of the model function which is in turn the name of the module.
`paras`	All other parameters that should be passed to the model function. i.e. model[[1]]$paras
`workEnv`	The environment name of the workflow call environment.

Value

A list of length 2 containing the model trained on all data and a data.frame which contains value, type, fold, lon, lat, predictions and then all environmental variables.

`subsetColumnsZoon` *subsetColumnsZoon*

Description

Extract a subset of columns from a data.frame, while preserving the user defined attributes of the parent data.frame.

Usage

```
subsetColumnsZoon(df, columns)
```

Arguments

`df`	The dataframe of which you'd like to subset columns from
`columns`	A vector of column names (character) of indices (numeric) which you'd like to keep

`summary.zoonWorkflow` *A function to summarize the output of a zoon workflow*

Description

Renders and opens an HTML report

Usage

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

Arguments

`object`	A zoonWorkflow object
`...`	currently ignored

Value

Path to HTML file. Associated images will be in the same directory.

`TransformCRS` *Change the CRS of occurrence data*

Description

Takes a dataframe returned by an occurrence module or a raster object from a covariate module and converts the CRS to lat/long so that everything works together.

Usage

```
TransformCRS(occurrence, ras_projection)
```

Arguments

`occurrence`	The output of an occurrence module
`ras_projection`	The projection of a covariate layer as a character (from projection())

Value

The same object as in occurrence, but with CRS changed as needed

`UKAirRas` *UK Air temperature raster layer.*

Description

This is an example environmental covariate raster layer. It is surface temperatures for the UK taken from NCEP

Format

Raster layer

Author(s)

Tim Lucas September 2014

Source

NCEP

`workflow` *Run a full workflow.*

Description

This is the main function of zoon. The arguments should specify at least five modules, at least one of each type. If modules do not have any arguments to be specific (or defaults are being used) then simply give the names of the module. If arguments are needed give the modules in the form of a function e.g. occurrence = AModule(para1 = 2, para2 = 'detail')

Usage

```
workflow(
  occurrence,
  covariate,
  process,
  model,
  output,
  forceReproducible = FALSE
)
```

Arguments

`occurrence`	Occurrence module to be used.
`covariate`	Covariate module to be used.
`process`	Process module to be used.
`model`	SDM model module to be used.
`output`	Output module to be used.
`forceReproducible`	Logical whether to force zoon to collect modules from the online repo. This ensure the analysis is reproducible.

Value

A list with the results of each module and a copy of the code used to execute the workflow. If the workflow fails a partial list is saved to a temporary file for debugging.

Examples

```
# run a workflow, using the logistic regression model
## Not run:

work1 <- workflow(occurrence = UKAnophelesPlumbeus,
                 covariate = UKAir,
                 process = Background(n = 70),
                 model = LogisticRegression,
                  output = SameTimePlaceMap)

str(work1, 1)

work2 <- workflow(UKAnophelesPlumbeus,
                   UKAir,
                   OneHundredBackground,
                   RandomForest,
                   PrintMap)


## End(Not run)
```

zoon — *Zoon: A package for comparing multiple SDM models, good model diagnostics and better reproducibility*

Description

Zoon: A package for comparing multiple SDM models, good model diagnostics and better reproducibility

ZoonCitation — *ZoonCitation*

Description

How to cite Zoon Modules in publications

Usage

```
ZoonCitation(ModuleName)
```

Arguments

`ModuleName`	string giving the name of the module

Value

A `zoonCitation` object - A list of elements that make up the citation

See Also

`print.zoonCitation`

ZoonFigshare *ZoonFigshare*

Description

This function uploads a zoon workflow object to Figshare. To share your workflow with the community please set your workflow to public on figshare after using this function. If your workflow is made public it will automatically appear on the Zoon website.

Usage

```
ZoonFigshare(
  zoonWorkflow,
  title = "My Zoon Workflow",
  description = "zoon workflow",
  authors = "zoon",
  categories = "SDM",
  tags = "zoon"
)
```

Arguments

`zoonWorkflow`	A zoonWorkflow object as returned by the function `Workflow`
`title`	String giving the title of the workflow
`description`	String describing the workflow
`authors`	Character vector of full authors names
`categories`	Character vector of figshare categories e.g. ecology.
`tags`	Character vector of searchable tags.

ZoonModel *ZoonModel*

Description

module developer tool: Create a Zoon model object

Usage

```
ZoonModel(model, code, packages)
```

Arguments

`model`	a fitted model object to be used for making predictions
`code`	code to make predictions from `model` object to a dataframe `newdata` containing new covariate observations. The code must use the objects named `model` and `newdata` and no other objects and must return a numeric vector, with the same length as the number of rows in `newdata` giving predictions on the response scale (e.g. probabilities of presence).
`packages`	a character vector giving the names of packages needed to run the code zoon-Model a `zoonModel` object

Details

This function is only intended to be used when developing new modules, not for running zoon workflows. Given a `zoonModel` object returned by a model module using the function `ZoonModel`, make a prediction to a new dataframe. For an example, see the source code for the module `InteractiveMap`.

Value

an object of class `zoonModel` containing all of the information and code required to make predictions, using the function `ZoonPredict`

`ZoonModuleParse` *Parse a module file to read roxygen tags*

Description

This is a stand in for the parse_file function in roxygen which is not exported.

Usage

```
ZoonModuleParse(modulePath)
```

Arguments

`modulePath`	The path to a zoon module .R script

Value

Named list of roxygen tags

`ZoonPredict` *ZoonPredict*

Description

module developer tool: Predict from a ZoonModel object

Usage

```
ZoonPredict(zoonModel, newdata)
```

Arguments

`zoonModel`	a `zoonModel` object
`newdata`	a dataframe containing data to predict to.

Details

This function is only intended to be used when developing new modules, not for running zoon workflows. Given a `zoonModel` object returned by a model module using the function `ZoonModel`, make a prediction to a new dataframe. Values returned must be on the response scale (e.g. probabilities of presence). For an example, see the source code for the module `mgcv`.

Index