Package: enmSdmX (via r-universe)

October 22, 2026

Type Package

Title Species Distribution Modeling and Ecological Niche Modeling

Version 1.2.12

Date 2025-03-289

Description Implements species distribution modeling and ecological niche modeling, including: bias correction, spatial cross-validation, model evaluation, raster interpolation, biotic ``velocity'' (speed and direction of movement of a ``mass'' represented by a raster), interpolating across a time series of rasters, and use of spatially imprecise records. The heart of the package is a set of ``training'' functions which automatically optimize model complexity based number of available occurrences. These algorithms include MaxEnt, MaxNet, boosted regression trees/gradient boosting machines, generalized additive models, generalized linear models, natural splines, and random forests. To enhance interoperability with other modeling packages, no new classes are created. The package works with 'PROJ6' geodetic objects and coordinate reference systems.

Depends R (>= 4.0.0)

Imports AICcmodavg, boot, cowplot, data.table, doParallel, DT, foreach, gbm, ggplot2, graphics, ks, maxnet, methods, mgcv, omnibus, parallel, predicts, ranger, rJava, scales, sf, shiny, sp, statisfactory, stats, terra, utils

LazyData true

LazyLoad yes

URL https://github.com/adamlilith/enmSdmX

BugReports https://github.com/adamlilith/enmSdmX/issues

Encoding UTF-8

License MIT + file LICENSE

RoxygenNote 7.3.2

NeedsCompilation no

Author Adam B. Smith [cre, aut]
(<https://orcid.org/0000-0002-6420-1659>)

Maintainer Adam B. Smith <adam.smith@mobot.org>

Repository https://cran.r-universe.dev

Date/Publication 2025-03-29 05:30:08 UTC

RemoteUrl https://github.com/cran/enmSdmX

RemoteRef HEAD

RemoteSha 8e4aeddcbd41dce6d10e621f0406506ab4b3f292

Contents

bioticVelocity . . . 3
canada . . . 15
compareResponse . . . 16
coordImprecision . . . 18
countPoints . . . 20
crss . . . 21
customAlbers . . . 22
decimalToDms . . . 24
dmsToDecimal . . . 24
elimCellDuplicates . . . 25
evalAUC . . . 26
evalContBoyce . . . 28
evalMultiAUC . . . 30
evalThreshold . . . 31
evalThresholdStats . . . 33
evalTjursR2 . . . 35
evalTSS . . . 37
extentToVect . . . 38
geoFold . . . 39
geoFoldContrast . . . 41
geoThin . . . 43
getCRS . . . 45
getValueByCell . . . 46
globalx . . . 47
interpolateRasts . . . 48
lemurs . . . 52
longLatRasts . . . 52
mad0 . . . 53
mad1 . . . 54
madClim . . . 55
madClim2030 . . . 55
madClim2050 . . . 56
madClim2070 . . . 57

madClim2090 . . . 57
modelSize . . . 58
nearestEnvPoints . . . 59
nearestGeogPoints . . . 63
nicheOverlapMetrics . . . 69
plotExtent . . . 70
predictEnmSdm . . . 71
predictMaxEnt . . . 77
predictMaxNet . . . 83
responseCurves . . . 89
sampleRast . . . 95
spatVectorToSpatial . . . 96
squareCellRast . . . 97
summaryByCrossValid . . . 98
trainBRT . . . 103
trainByCrossValid . . . 109
trainESM . . . 115
trainGAM . . . 119
trainGLM . . . 126
trainMaxEnt . . . 134
trainMaxNet . . . 141
trainNS . . . 147
trainRF . . . 154
troubleshooting_parallel_operations . . . 161
weightByDist . . . 162

Index **164**

`bioticVelocity` *Velocity of shifts in densities across a series of rasters*

Description

Calculates metrics of "movement" of cell densities across a time series of rasters. Rasters could represent, for example, the probability of presence of a species through time. In this case, velocities would indicate rates and directions of range shift. The simplest metric is movement of the density-weighted centroid (i.e., range "center"), but many more are available to provide a nuanced indicator of velocity. See `Details` for the types of metrics that can be calculated.

Usage

```
bioticVelocity(
  x,
  times = NULL,
  atTimes = NULL,
  elevation = NULL,
  metrics = c("centroid", "nsCentroid", "ewCentroid", "nCentroid", "sCentroid",
    "eCentroid", "wCentroid", "nsQuants", "ewQuants", "similarity", "summary"),
```

```
  quants = c(0.05, 0.1, 0.5, 0.9, 0.95),
  onlyInSharedCells = FALSE,
  cores = 1,
  warn = TRUE,
  longitude = NULL,
  latitude = NULL,
  paths = NULL,
  ...
)
```

Arguments

x	Either a SpatRaster or a 3-dimensional array. Values *should really be* be either NA or >= 0. • If x is a SpatRaster, then each layer is assumed to represent a time slice. Rasters *must* be in an equal-area projection. They must also be ordered temporally, with the raster "on top" assumed to represent the starting time. • If x is an array then each "layer" in the third dimension is assumed to represent a map at a particular time slice in an equal-area projection. Note that if this is an array you should probably also specify the arguments longitude and latitude.
times	Numeric vector with the same number of layers in x or NULL (default). This specifies the time represented by each layer in x from beginning of the time series (top layer) to the end (bottom layer). Times *must* appear in sequential order. For example, if time periods are 24 kybp, 23 kybp, 22 kybp, use c(-24, -23, -22), not c(24, 23, 22). If NULL (default), values are assigned starting at 1 and ending at the total number of layers in x.
atTimes	Numeric, values of times across which to calculate biotic velocity. You can use this to calculate biotic velocities across selected time periods (e.g., just the first and last time periods). Note that atTimes must be the same as or a subset of times. The default is NULL, in which case velocity is calculated across all time slices (i.e., between times 1 and 2, 2 and 3, 3 and 4, etc.).
elevation	Either NULL (default) or a raster or matrix representing elevation. If this is supplied, changes in elevation are incorporated into all velocity and speed metrics. Additionally, you can also calculate the metrics elevCentrioid and elevQuants.
metrics	Biotic velocity metrics to calculate (default is to calculate them all). All metrics ignore NA cells in x. Here, "starting time period" represents one layer in x and "end time period" the next layer. • centroid: Speed of mass-weighted centroid (directionless). • nsCentroid or ewCentroid: Velocity in the north-south or east-west directions of the mass-weighted centroid. • nCentroid, sCentroid, eCentroid, and wCentroid: Speed of mass-weighted centroid of the portion of the raster north/south/east/west of the landscape-wide weighted centroid of the starting time period. • nsQuants or ewQuants: Velocity of the location of the Qth quantile of mass in the north-south or east-west directions. The quantiles can be specified in

quants. For example, this could be the movement of the 5th, 50th, and 95th quantiles of population size going from south to north. The 0th quantile would measure the velocity of the southernmost or easternmost cell(s) with values >0, and the 100th quantile the northernmost or westernmost cell(s) with non-zero values.

- similarity: Metrics of similarity between each time period. Some of these make sense only for cases where values in x are in the range [0, 1], but not if some values are outside this range. See nicheOverlapMetrics for more details. The metrics are:
  - Simple mean difference
  - Mean absolute difference
  - Root-mean squared difference
  - Expected Fraction of Shared Presences or ESP (Godsoe, W. 2014. *Ecography* 37:130-136 doi:10.1111/j.16000587.2013.00403.x)
  - D statistic (Schoener, T.W. 1968. *Ecology* 49:704-726. doi:10.2307/1935534)
  - I statistic (Warren, D.L., et al. 2008. *Evolution* 62:2868-2883 doi:10.1111/j.15585646.2008.00482.x)
  - Pearson correlation
  - Spearman rank correlation
- summary: This calculates a series of measures for each "starting time period" raster. None of these are measures of velocity:
  - Mean: Mean value across all cells.
  - Sum: Total across all cells.
  - Quantiles: *Q*th quantile values across all cells. Quantiles are provided through argument quants.
  - Prevalence: Number of cells with values > 0.
- elevCentroid: Velocity of the centroid of mass in elevation (up or down). A raster or matrix must be supplied to argument elevation.
- elevQuants: Velocity of the *Q*th quantile of mass in elevation (up or down). The quantiles to be evaluated are given by quants. The lowest elevation with mass >0 is the 0th quantile, and the highest elevation with mass >0 is the 100th. Argument elevation must be supplied.

quants — Numeric vector indicating the quantiles at which biotic velocity is calculated for the "quant" and "Quants" metrics. Default quantiles to calculate are c(0.1, 0.9).

onlyInSharedCells — If TRUE, calculate biotic velocity using only those cells that are not NA in the start *and* end of each time period. This is useful for controlling for shifting land mass due to sea level rise, for example, when calculating biotic velocity for an ecosystem or a species. The default is FALSE, in which case velocity is calculated using all cells in each time period, regardless of whether some become NA or change from NA to not NA.

cores — Positive integer. Number of processor cores to use. Note that if the number of time steps at which velocity is calculated is small, using more cores

may not always be faster. If you have issues when cores > 1, please see the troubleshooting_parallel_operations guide.

warn — Logical, if TRUE (default) then display function-specific warnings.

longitude — Numeric matrix or NULL (default):

- If x is a SpatRaster, then this is ignored (longitude is ascertained directly from the rasters, which *must* be in equal-area projection for velocities to be valid).
- If x is an array and longitude is NULL (default), then longitude will be ascertained from column numbers in x and velocities will be in arbitrary spatial units (versus, for example, meters). Alternatively, this can be a two-dimensional matrix whose elements represent the longitude coordinates of the centers of cells of x. The matrix must have the same number of rows and columns as x. Coordinates must be from an equal-area projection for results to be valid.

latitude — Numeric matrix or NULL (default):

- If x is a SpatRaster, then this is ignored (latitude is obtained directly from the rasters, which *must* be in equal-area projection for velocities to be valid).
- If x is an array and latitude is NULL (default), then latitude will be obtained from row numbers in x and velocities will be in arbitrary spatial units (versus, for example, meters). Alternatively, this can be a two-dimensional matrix whose elements represent the latitude coordinates of the centers of cells of x. The matrix must have the same number of rows and columns as x. Coordinates must be from an equal-area projection for results to be valid.

paths — This is used internally and rarely (never?) needs to be defined by a user (i.e., leave it as NULL). Valid values are a character vector or NULL (default). If a character vector, it should give the values used by .libPaths.

... — Other arguments (not used).

Details

Attention:

This function may yield erroneous velocities if the region of interest is near or spans a pole or the international date line. Results using the "Quant" and "quant" metrics may be somewhat counterintuitive if just one cell is >0, or one row or column has the same values with all other values equal to 0 or NA because defining quantiles in these situations is not intuitive. Results may also be counterintuitive if some cells have negative values because they can "push" a centroid away from what would seem to be the center of mass as assessed by visual examination of a map.

Note:

For the nsQuants and ewQuants metrics it is assumed that the latitude/longitude assigned to a cell is at its exact center. For calculating the position of a quantile, density is interpolated linearly from one cell center to the center of the adjacent cell. If a desired quantile does not fall exactly on the cell center, it is calculated from the interpolated values. For quantiles that fall south/westward of the first row/column of cells, the cell border is assumed to be at 0.5 * cell length south/west of the cell center.

Value

A data frame with biotic velocities and related values. Fields are as follows:

- `timeFrom`: Start time of interval
- `timeTo`: End time of interval
- `timeMid`: Time point between `timeFrom` and `timeTo`
- `timeSpan`: Duration of interval

Depending on `metrics` that are specified, additional fields are as follows. All measurements of velocity are in distance units (typically meters) per time unit (which is the same as the units used for `times` and `atTimes`). For example, if the rasters are in an Albers equal-area projection and `times` are in years, then the output will be meters per year.

- If `metrics` has `'centroid'`: Columns named `centroidVelocity`, `centroidLong`, `centroidLat` – Speed of weighted centroid, plus its longitude and latitude (in the `timeTo` period of each time step). Values are always >= 0.
- If `metrics` has `'nsCentroid'`: Columns named `nsCentroid` and `nsCentroidLat` – Velocity of weighted centroid in north-south direction, plus its latitude (in the `timeTo` period of each time step). Positive values connote movement north, and negative values south.
- If `metrics` has `'ewCentroid'`: `ewCentroid` and `ewCentroidLong` – Velocity of weighted centroid in east-west direction, plus its longitude (in the `timeTo` period of each time step). Positive values connote movement east, and negative values west.
- If `metrics` has `'nCentroid'`, `'sCentroid'`, `'eCentroid'`, and/or `'wCentroid'`: Columns named `nCentroidVelocity` and `nCentroidAbund`, `sCentroid` and `sCentroidAbund`, `eCentroid` and `eCentroidAbund`, and/or `wCentroid` and `wCentroidAbund` – Speed of weighted centroid of all cells that fall north, south, east, or west of the landscape-wide centroid, plus a column indicating the total weight (abundance) of all such populations. Values are always >= 0.
- If `metrics` contains any of `nsQuants` or `ewQuants`: Columns named `nsQuantVelocity_quant`Q and `nsQuantLat_quant`Q, or `ewQuantVelocity_quant`Q and `ewQuantLat_quant`Q: Velocity of the Qth quantile weight in the north-south or east-west directions, plus the latitude or longitude thereof (in the `timeTo` period of each time step). Quantiles are cumulated starting from the south or the west, so the 0.05th quantile, for example, is in the far south or west of the range and the 0.95th in the far north or east. Positive values connote movement north or east, and negative values movement south or west.
- If `metrics` contains `similarity`, metrics of similarity are calculated for each pair of successive landscapes, defined below as `x1` (raster in `timeFrom`) and `x2` (raster in `timeTo`), with the number of shared non-`NA` cells between them being `n`:
  - A column named `simpleMeanDiff`: `sum(x2 - x1, na.rm = TRUE) / n`
  - A column named `meanAbsDiff`: `sum(abs(x2 - x1), na.rm = TRUE) / n`
  - A column named `rmsd` (root-mean square difference): `sqrt(sum((x2 - x1)^2, na.rm = TRUE)) / n`
  - A column named `godsoeEsp`: `1 - sum(2 * (x1 * x2), na.rm=TRUE) / sum(x1 + x2, na.rm = TRUE)`, values of 1 ==> maximally similar, 0 ==> maximally dissimilar.
  - A column named `schoenersD`: `1 - (sum(abs(x1 - x2), na.rm = TRUE) / n)`, values of 1 ==> maximally similar, 0 ==> maximally dissimilar.

  - A column named `warrensI`: `1 - sqrt(sum((sqrt(x1) - sqrt(x2))^2, na.rm = TRUE) / n)`, values of 1 ==> maximally similar, 0 ==> maximally dissimilar.
  - A column named `cor`: Pearson correlation between `x1` and `x2`.
  - A column named `rankCor`: Spearman rank correlation between `x1` and `x2`.
- If `metrics` contains `elevCentroid`: Columns named `elevCentroidVelocity` and `elevCentroidElev` – Velocity of the centroid in elevation (up or down) and the elevation in the "to" timestep. Positive values of velocity connote movement upward, and negative values downward.
- If `metrics` contains `elevQuants`: Columns named `elevQuantVelocity_quant`Q and `elevQuantVelocityElev_quan` – Velocity of the Nth quantile of mass in elevation (up or down) and the elevation of this quantile in the "to" timestep. Positive values of velocity connote movement upward, and negative values downward.
- If `metrics` contains `summary`:
  - A column named `propSharedCellsNotNA`: Proportion of cells that are not `NA` in *both* the "from" and "to" time steps. The proportion is calculated using the total number of cells in a raster as the denominator (i.e., not total number of cells across two rasters).
  - Columns named `timeFromPropNotNA` and `timeToPropNotNA`: Proportion of cells in the "from" time and "to" steps that are not `NA`.
  - A column named `mean`: Mean weight in `timeTo` time step. In the same units as the values of the cells.
  - Columns named `quantile_quant`Q: The Qth quantile(s) of weight in the `timeTo` time step. In the same units as the values of the cells.
  - A column named `prevalence`: Proportion of non-`NA` cells with weight >0 in the `timeTo` time step relative to all non-`NA` cells. Unitless.

Examples

```
# NB These examples can take a few minutes to run.
# To illustrate calculation and interpretation of biotic velocity,
# we will calibrate a SDM for the Red-Bellied Lemur and project
# the model to the present and successive future climates. The time series
# of rasters is then used to calculate biotic velocity.

library(sf)
library(terra)

### process environmental rasters
#################################

# get rasters
rastFile <- system.file('extdata/madClim.tif', package='enmSdmX')
madClim <- rast(rastFile)

rastFile <- system.file('extdata/madClim2030.tif', package='enmSdmX')
madClim2030 <- rast(rastFile)

rastFile <- system.file('extdata/madClim2050.tif', package='enmSdmX')
madClim2050 <- rast(rastFile)

rastFile <- system.file('extdata/madClim2070.tif', package='enmSdmX')
```

```
madClim2070 <- rast(rastFile)

rastFile <- system.file('extdata/madClim2090.tif', package='enmSdmX')
madClim2090 <- rast(rastFile)

# The bioticVelocity() function needs rasters to be in equal-area
# projection, so we will project them here.
madAlbers <- getCRS('madAlbers') # Albers projection for Madagascar
madClim <- project(madClim, madAlbers)
madClim2030 <- project(madClim2030, madAlbers)
madClim2050 <- project(madClim2050, madAlbers)
madClim2070 <- project(madClim2070, madAlbers)
madClim2090 <- project(madClim2090, madAlbers)

# Coordinate reference systems:
wgs84 <- getCRS('WGS84') # WGS84
madAlbers <- getCRS('madAlbers') # Madagascar Albers

# lemur occurrence data
data(lemurs)
occs <- lemurs[lemurs$species == 'Eulemur fulvus', ]
occs <- vect(occs, geom=c('longitude', 'latitude'), crs=wgs84)
occs <- project(occs, madAlbers)

# eliminate cell duplicates
occs <- elimCellDuplicates(occs, madClim)

# extract environment at occurrences
occEnv <- extract(madClim, occs, ID = FALSE)
occEnv <- occEnv[complete.cases(occEnv), ]

# create background sites (using just 1000 to speed things up!)
bgEnv <- terra::spatSample(madClim, 3000)
bgEnv <- bgEnv[complete.cases(bgEnv), ]
bgEnv <- bgEnv[sample(nrow(bgEnv), 1000), ]

# collate occurrences and background sites
presBg <- data.frame(
   presBg = c(
      rep(1, nrow(occEnv)),
      rep(0, nrow(bgEnv))
   )
)

env <- rbind(occEnv, bgEnv)
env <- cbind(presBg, env)

### calibrate model
###################

predictors <- c('bio1', 'bio12')

# MaxEnt
```

```
mx <- trainMaxEnt(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 cores = 2
)

### project to present and future climate
########################################

predPresent <- predictEnmSdm(mx, madClim)
pred2030 <- predictEnmSdm(mx, madClim2030)
pred2050 <- predictEnmSdm(mx, madClim2050)
pred2070 <- predictEnmSdm(mx, madClim2070)
pred2090 <- predictEnmSdm(mx, madClim2090)

plot(predPresent, main = 'Present Suitability')

# plot change in suitability between present and 2090s
delta <- pred2090 - predPresent
plot(delta, main = 'Change in Suitability')

### calculate biotic velocity
#############################

series <- c(
 predPresent,
 pred2030,
 pred2050,
 pred2070,
 pred2090
)

names(series) <- c('present', 't2030', 't2050', 't2070', 't2090')
plot(series)

times <- c(1985, 2030, 2050, 2070, 2090)
quants <- c(0.10, 0.90)

bv <- bioticVelocity(
 x = series,
 times = times,
 quants = quants,
 cores = 2
)

bv

### centroid velocities

# centroid (will always be >= 0)
# fastest centroid movement around 2060
```

```
plot(bv$timeMid, bv$centroidVelocity, type = 'l',
  xlab = 'Year', ylab = 'Speed (m / y)', main = 'Centroid Speed')

# velocity northward/southward through time
# shows northward shift because always positive, fastest around 2060
plot(bv$timeMid, bv$nsCentroidVelocity, type = 'l',
  xlab = 'Year', ylab = 'Velocity (m / y)', main = 'Centroid N/S Velocity')

# velocity eastward (positive)/westward (negative) through time
# movement eastward (positive) first, then westward (negative)
plot(bv$timeMid, bv$ewCentroidVelocity, type = 'l',
  xlab = 'Year', ylab = 'Velocity (m / y)', main = 'Centroid E/W Velocity')

### map of centroid location through time
# shows centroid moves slightly northward through time
plot(delta, main = 'Centroid Location &\nChange in Suitability')
points(bv$centroidLong[1], bv$centroidLat[1], pch = 1)
points(bv$centroidLong[4], bv$centroidLat[4], pch = 16)
lines(bv$centroidLong, bv$centroidLat)
legend(
  'bottomright',
  legend = c(
    'start (~1985)',
 'stop (~2090)',
 'trajectory'
  ),
  pch = c(1, 16, NA),
  lwd = c(NA, NA, 1)
)

### velocities of portions of range north/south/east/west of centroid
# positive ==> northward shift
# negative ==> southward shift

# portion of range north of centroid
# shows northward expansion because always positive
plot(bv$timeMid, bv$nCentroidVelocity, type = 'l',
  xlab = 'Year', ylab = 'Velocity (m / y)',
  main = 'Northern Part of Range')

# portion of range south of centroid
# shows northward contraction because always positive
plot(bv$timeMid, bv$sCentroidVelocity, type = 'l',
  xlab = 'Year', ylab = 'Velocity (m / y)',
  main = 'Southern Part of Range')

# portion of range east of centroid
# shows eastern portion moves farther east
plot(bv$timeMid, bv$eCentroidVelocity, type = 'l',
  xlab = 'Year', ylab = 'Velocity (m / y)',
  main = 'Eastern Part of Range')

# portion of range west of centroid
```

```
# shows western portion moves east
plot(bv$timeMid, bv$wCentroidVelocity, type = 'l',
  xlab = 'Year', ylab = 'Velocity (m / y)',
  main = 'Western Part of Range')

### velocities of range margins

# from south to north, 10th and 90th quantiles of density
# positive ==> northward shift
# negative ==> southward shift
# shows both northern and southern range margins shift northward
# because always positive... northern margin shift usually slower
ylim <- range(bv$nsQuantVelocity_quant0p1, bv$nsQuantVelocity_quant0p9)

plot(bv$timeMid, bv$nsQuantVelocity_quant0p1, type = 'l', ylim = ylim,
  xlab = 'Year', ylab = 'Velocity (m / y)',
  main = 'Northern/Southern Range Margins')
lines(bv$timeMid, bv$nsQuantVelocity_quant0p9, lty = 'dashed')
legend(
  'bottomright',
  legend = c('Southern Margin', 'Northern Margin'),
  lty = c('solid', 'dashed')
)

# from east to west, 10th and 90th quantiles of density
# positive ==> eastward shift
# negative ==> westward shift
ylim <- range(bv$ewQuantVelocity_quant0p1, bv$ewQuantVelocity_quant0p9)

plot(bv$timeMid, bv$ewQuantVelocity_quant0p1, type = 'l', ylim = ylim,
  xlab = 'Year', ylab = 'Velocity (m / y)',
  main = 'Eastern/Western Range Margins')
lines(bv$timeMid, bv$ewQuantVelocity_quant0p9, lty = 'dashed')
legend(
  'bottomright',
  legend = c('Eastern Margin', 'Western Margin'),
  lty = c('solid', 'dashed')
)


### summary statistics

# mean density across cells through time
plot(bv$timeMid, bv$mean, type = 'l',
  xlab = 'Year', ylab = 'Mean Density',
  main = 'Mean Density')

# sum of density across cells through time
plot(bv$timeMid, bv$sum, type = 'l',
  xlab = 'Year', ylab = 'Sum of Density',
  main = 'Sum of Density')

### change metrics
```

```
# average change in suitability from one time period to next
# shows average conditions getting worse
plot(bv$timeMid, bv$simpleMeanDiff, type = 'l',
  xlab = 'Year', ylab = 'Mean Change in Suitability')

# average absolute change in suitability from one time period to next
# shows average absolute change declining
plot(bv$timeMid, bv$meanAbsDiff, type = 'l',
  xlab = 'Year', ylab = 'Mean Absolute Change in Suitability')

# root-mean square difference from one time period to the next
# shows difference between successive rasters declines through time
plot(bv$timeMid, bv$rmsd, type = 'l',
  xlab = 'Year', ylab = 'RMSD')

### raster similarity
# most indicate that successive rasters are similar through time
ylim <- range(bv$godsoeEsp, bv$schoenerD, bv$warrenI, bv$cor, bv$warrenI)
plot(bv$timeMid, bv$godsoeEsp, type = 'l', lty = 1, col = 1,
  xlab = 'Year', ylab = 'Raster similarity', ylim = ylim)
lines(bv$timeMid, bv$schoenerD, lty = 2, col = 2)
lines(bv$timeMid, bv$warrenI, lty = 3, col = 3)
lines(bv$timeMid, bv$cor, lty = 4, col = 4)
lines(bv$timeMid, bv$rankCor, lty = 5, col = 5)

legend(
  'right',
  legend = c(
    'Godsoe\'s ESP',
    'Schoener\'s D',
    'Warren\'s I',
    'Correlation',
    'Rank Correlation'
  ),
  col = 1:5,
  lty = 1:5
)


# values of 10th and 90th quantiles across cells through time
# shows most favorable cells becoming less favorable
# least favorable cells remain mainly unchanged
ylim <- range(bv$quantile_quant0p1, bv$quantile_quant0p9)

plot(bv$timeMid, bv$quantile_quant0p1, type = 'l', ylim = ylim,
  xlab = 'Year', ylab = 'Quantile Value',
  main = 'Quantiles across Cells')
lines(bv$timeMid, bv$quantile_quant0p9, lty = 'dashed')

legend(
  'topright',
  legend = c('10th quantile', '90th quantile'),
```

```
  lty = c('solid', 'dashed')
)

### map of northern/southern range margins through time

# range of longitude shown in plot
madExtent <- ext(madClim)
xExtent <- as.vector(madExtent)[1:2]

plot(predPresent, main = 'North/South Range Margin Location')
lines(c(xExtent[1], xExtent[2]),
  c(bv$nsQuantLat_quant0p9[1], bv$nsQuantLat_quant0p9[1]))
lines(c(xExtent[1], xExtent[2]),
  c(bv$nsQuantLat_quant0p9[2], bv$nsQuantLat_quant0p9[2]), lty = 'dashed')
lines(c(xExtent[1], xExtent[2]),
  c(bv$nsQuantLat_quant0p9[3], bv$nsQuantLat_quant0p9[3]), lty = 'dotdash')
lines(c(xExtent[1], xExtent[2]),
  c(bv$nsQuantLat_quant0p9[4], bv$nsQuantLat_quant0p9[4]), lty = 'dotted')

lines(c(xExtent[1], xExtent[2]),
  c(bv$nsQuantLat_quant0p1[1], bv$nsQuantLat_quant0p1[1]))
lines(c(xExtent[1], xExtent[2]),
  c(bv$nsQuantLat_quant0p1[2], bv$nsQuantLat_quant0p1[2]), lty = 'dashed')
lines(c(xExtent[1], xExtent[2]),
  c(bv$nsQuantLat_quant0p1[3], bv$nsQuantLat_quant0p1[3]), lty = 'dotdash')
lines(c(xExtent[1], xExtent[2]),
  c(bv$nsQuantLat_quant0p1[4], bv$nsQuantLat_quant0p1[4]), lty = 'dotted')

legend(
  'bottomright',
  legend = c(
    '1980s',
 '2030s',
 '2050s',
 '2070s',
 '2090s'
  ),
  lty = c('solid', 'dashed', 'dotdash', 'dotted')
)

### map of eastern/western range margins through time

# range of longitude shown in plot
madExtent <- ext(madClim)
yExtent <- as.vector(madExtent)[3:4]

plot(predPresent, main = 'North/South Range Margin Location')
lines(c(bv$ewQuantLong_quant0p9[1], bv$ewQuantLong_quant0p9[1]),
  c(yExtent[1], yExtent[2]))
lines(c(bv$ewQuantLong_quant0p9[2], bv$ewQuantLong_quant0p9[2]),
  c(yExtent[1], yExtent[2]), lty = 'dashed')
lines(c(bv$ewQuantLong_quant0p9[3], bv$ewQuantLong_quant0p9[3]),
  c(yExtent[1], yExtent[2]), lty = 'dotdash')
```

```
lines(c(bv$ewQuantLong_quant0p9[4], bv$ewQuantLong_quant0p9[4]),
  c(yExtent[1], yExtent[2]), lty = 'dotted')

lines(c(bv$ewQuantLong_quant0p1[1], bv$ewQuantLong_quant0p1[1]),
  c(yExtent[1], yExtent[2]))
lines(c(bv$ewQuantLong_quant0p1[2], bv$ewQuantLong_quant0p1[2]),
  c(yExtent[1], yExtent[2]), lty = 'dashed')
lines(c(bv$ewQuantLong_quant0p1[3], bv$ewQuantLong_quant0p1[3]),
  c(yExtent[1], yExtent[2]), lty = 'dotdash')
lines(c(bv$ewQuantLong_quant0p1[4], bv$ewQuantLong_quant0p1[4]),
  c(yExtent[1], yExtent[2]), lty = 'dotted')

legend(
  'bottomright',
  legend = c(
    '1980s',
 '2030s',
 '2050s',
 '2070s',
 '2090s'
  ),
  lty = c('solid', 'dashed', 'dotdash', 'dotted')
)
```

---

canada *Vector outline of Canada*

Description

This `SpatVector` represents the outline of Canada in WGS84 (unprojected) coordinates. This is the "low resolution" (less accurate) version from GADM.

Format

An object of class `'SpatVector'`.

Source

Database of Global Administrative Areas (GADM)

Examples

```
library(terra)
canFile <- system.file('extdata', 'canada_level0_gadm41.gpkg', package='enmSdmX')
canada <- vect(canFile)
plot(canada)
```

compareResponse *Compare two response curves along one or more predictors*

Description

This function calculates a suite of metrics reflecting of niche overlap for two response curves. Response curves are predicted responses of a uni- or multivariate model along a single variable. Depending on the user-specified settings the function calculates these values either at each pair of values of pred1 and pred2 *or* along a smoothed version of pred1 and pred2.

Usage

```
compareResponse(
  pred1,
  pred2,
  data,
  predictor = names(data),
  adjust = FALSE,
  gap = Inf,
  smooth = FALSE,
  smoothN = 1000,
  smoothRange = c(0, 1),
  graph = FALSE,
  ...
)
```

Arguments

pred1	Numeric vector. Predictions from first model along data (one value per row in data).
pred2	Numeric vector. Predictions from second model along data (one value per row in data).
data	Data frame or matrix corresponding to pred1 and pred2.
predictor	Character vector. Name(s) of predictor(s) for which to calculate comparisons. These must appear as column names in data.
adjust	Logical. If TRUE then subtract the mean of pred1 from pred1 and the mean of pred2 from pred2 before analysis. Useful for comparing the shapes of curves while controlling for different elevations (intercepts).
gap	Numeric >0. Proportion of range of predictor variable across which to assume a gap exists. Calculation of areaAbsDiff will ignore gaps wide than this. To ensure the entire range of the data is included set this equal to Inf (default).
smooth	Logical. If TRUE then the responses are first smoothed using loess() then compared at smoothN values along each predictor. If FALSE, then comparisons are conducted at the raw values pred1 and pred2.

smoothN	NULL or positive integer. Number of values along "pred" at which to calculate comparisons. Only used if smooth is TRUE. If NULL, then comparisons are calculated at each value in data. If a number, then comparisons are calculated at smoothN values of data[, pred] that cover the range of data[, pred].
smoothRange	2-element numeric vector or NULL. If smooth is TRUE, then force loess predictions < smoothRange[1] to equal smoothRange[1] and predictions > smoothRange[2] to equal smoothRange[2]. Ignored if NULL.
graph	Logical. If TRUE then plot predictions.
...	Arguments to pass to functions like sum() (for example, na.rm=TRUE) and to overlap() (for example, w for weights). Note that if smooth = TRUE, then passing an argument called w will likely cause a warning and make results circumspect *unless* weights are pre-calculated for each of the smoothN points along a particular predictor.

Value

Either a data frame (if smooth = FALSE or a list object with the smooth model plus a data frame (if smooth = TRUE) . The data frame represents metrics comparing response curves of pred1 and pred2:

- predictor Predictor for which comparison was made
- n Number of values of predictor at which comparison was calculated
- adjust adjust argument.
- smooth smooth argument.
- meanDiff Mean difference between predictions of pred1 and pred2 (higher ==> more different).
- meanAbsDiff Mean absolute value of difference between predictions of pred1 and pred2 (higher ==> more different).
- areaAbsDiff Sum of the area between curves predicted by pred1 and pred2, standardized by total potential area between the two curves (i.e., the area available between the minimum and maximum prediction along the minimum and maximum values of the predictor) (higher ==> more different).
- d Schoener's *D*
- i Hellinger's *I* (adjusted to have a range [0, 1])
- esp Godsoe's ESP
- cor Pearson correlation between predictions of pred1 and pred2.
- rankCor Spearman rank correlation between predictions of pred1 and pred2.

References

Warren, D.L., Glor, R.E., and Turelli, M. 2008. Environmental niche equivalency versus conservatism: Quantitative approaches to niche evolution. Evolution 62:2868-2883.

Warren, D.L., Glor, R.E., and Turelli, M. 2008. Erratum. Evolution 62:2868-2883.

Godsoe, W. 2014. Inferring the similarity of species distributions using Species Distribution Models. Ecography 37:130-136.

See Also

nicheOverlapMetrics

Examples

```
set.seed(123)
data <- data.frame(
 x1=seq(-1, 1, length.out=100),
 x2=seq(-1, 1, length.out=100) + rnorm(100, 0, 0.3)
)

pred1 <- 1 / (1 + exp(-(0.3 + 2 * (data$x1 - 0.2) -0.3 * data$x2)))
pred2 <- 1 / (1 + exp(-(-0 + 0.1 * data$x1 - 4 * data$x1^2 + 0.4 * data$x2)))

compareResponse(pred1, pred2, data, graph=TRUE)
compareResponse(pred1, pred2, data, smooth=TRUE, graph=TRUE)
compareResponse(pred1, pred2, data, adjust=TRUE, graph=TRUE)
```

---

coordImprecision *Calculate the precision of a geographic coordinate*

Description

This function calculates the imprecision of geographic coordinates due to rounded coordinate values. See *Details* for an explanation of how this is calculated.

Usage

```
coordImprecision(x, dms = FALSE, epsilon = 2)
```

Arguments

x	Spatial points represented as a SpatVector or sf object. Alternatively, this can also be a data frame or matrix, in which the first two columns must represent longitude and latitude (in that order). If x is a matrix or data frame, the coordinates are assumed to be unprojected (WGS84).
dms	Logical, if FALSE (default), it is assumed that the original format in which coordinate were reported is in decimal notation. If TRUE, then it will be calculated assuming the coordinate were originally in degrees-minutes-seconds format. If you do not know the original format, the less presumptive approach is to calculate coordinate imprecision twice with or without dm = TRUE, and use the larger of the two values.
epsilon	Zero or positive integer, number of digits to which to round seconds value if dms is TRUE. Ignored if dms is FALSE. This is used to accommodate inexact integer values when converting from DMS to decimal. For example, -108.932222 converted to DMS format is 108deg 55min 7.9992sec, but if epsilon = 2 then it would be converted to 108deg 55min 08sec.

Details

For coordinates originally reported in decimal notation, coordinate imprecision is *half* the distance between the two opposing corners on a bounding box whose size is based on the number of significant digits in the coordinates. This box is defined by 1) finding the maximum number of significant digits after the decimal in the longitude/latitude pair; 2) adding/subtracting 5 to the decimal place that falls just after this; and 3) calculating the distance between these points then dividing by 2. For example, if longitude is 82.37 and latitude 45.8 then the number of significant digits after the decimal place is 2 and 1, respectively so 2 is used on the assumption that latitude is measured to the nearest 100th of a degree. The precision is then the distance between the point pairs (82.37 - 0.005 = 82.365, 45.8 - 0.005 = 45.795) and (82.37 + 0.005 = 82.375, 45.8 + 0.005 = 45.805).

For coordinates originally reported in degree-minus-second (DMS) format, the bounding box is defined by adding/subtracting 0.5 units (degrees, minutes, or seconds, depending on the smallest non-zero unit reported) from the coordinate. For example, if longitude is 90deg 00min 00sec and latitude is 37deg 37min 37sec, then the bounding box will be defined by adding/subtracting 0.5 arcsec to the coordinates.

Value

Numeric values (by default in units of meters).

Examples

```
# coarse-precision cases
long <- c(45, 45.1, 45.1)
lat <-  c(45, 45.1, 45)
ll <- cbind(long, lat)
precision_m <- coordImprecision(ll)
cbind(ll, precision_m)

# fine-precision cases
long <- rep(45, 8)
lat <-  c(45, 45.1, 45.11, 45.111, 45.1111, 45.11111, 45.111111, 45.1111111)
ll <- cbind(long, lat)
precision_m <- coordImprecision(ll)
cbind(ll, precision_m)

# precision varies with latitude
long <- rep(45, 181)
lat <- seq(-90, 90)
ll <- cbind(long, lat)
precision_m <- coordImprecision(ll)
cbind(ll, precision_m)
plot(lat, precision_m / 1000, xlab='Latitude', ylab='Precision (km)')

# dateline/polar cases
long <- c(0, 180, 45, 45)
lat <-  c(45, 45, 90, -90)
ll <- cbind(long, lat)
```

```
precision_m <- coordImprecision(ll)
cbind(ll, precision_m)

# original coordinates in degrees-minutes-seconds format
longDD <- c(90, 90, 90, 90, 90, 90)
longMM <- c(0, 0, 0, 11, 11, 0)
longSS <- c(0, 0, 0, 0, 52, 52)
latDD <- c(38, 38, 38, 38, 38, 38)
latMM <- c(0, 37, 37, 37, 37, 0)
latSS <- c(0, 0, 38, 38, 38, 0)
longHemis <- rep('W', 6)
latHemis <- rep('N', 6)
longDec <- dmsToDecimal(longDD, longMM, longSS, longHemis)
latDec <- dmsToDecimal(latDD, latMM, latSS, latHemis)
decimal <- cbind(longDec, latDec)
(decImp <- coordImprecision(decimal))
(dmsImp <- coordImprecision(decimal, dms=TRUE))

# What if we do not know if coordinates were originally reported in
# decimal or degrees-minutes-seconds format? Most conservative option
# is to use maximum:
pmax(decImp, dmsImp)

# when longitude is negative and latitude is -90
long <- -45
lat <- -90
ll <- cbind(long, lat)
coordImprecision(ll)
```

`countPoints` *Number of points in a "spatial points" object*

Description

Returns the number of points in a `sf` or `SpatVector` object. This is typically done using either `length(x)` or `nrow(x)`, depending on whether the object in question has rows or not. This function helps in ambiguous cases, so users need not care if `nrow` or `length` is needed.

Usage

```
countPoints(x, byFeature = FALSE)
```

Arguments

`x`	Object of class `sf` or `SpatVector`
`byFeature`	If `FALSE`, return number of points for all features combined. If `TRUE`, report number of points per feature.

Value

Numeric.

Examples

```
library(sf)

# lemur occurrence data
data(lemurs)
wgs84 <- getCRS('WGS84')
occs <- lemurs[lemurs$species == 'Eulemur fulvus', ]
occs <- sf::st_as_sf(occs, coords=c('longitude', 'latitude'), crs=wgs84)

countPoints(occs)
```

---

crss *Coordinate reference systems (CRSs) and nicknames*

Description

A table of commonly-used coordinate reference systems, their nicknames, and WKT2 (well-known text) strings

Usage

```
data(crss)
```

Format

An object of class `data.frame`. This is a table with "named" coordinate referenbce systems and their well-known-text (WKT2) representation. It can be used as-is, or with `getCRS` to quickly get a WKT for a particular CRS. The fields are as:

- `long`: "Long" name of the CRS
- `short1` and `short2`: "Short" names of the CRS
- `region`: Region for which CRS is fit
- `projected`: Is the CRS projected or not?
- `projectionGeometry`: Type of projection (`NA`, 'cylindrical', 'conic', or 'planar')
- `datum`: Datum
- `type`: Either 'CRS' or 'data'. The former are proper CRSs, and the latter are those used by popular datasets.
- `wkt2`: WKT2 string.
- `notes`: Notes.

Examples

```
data(crss)
getCRS('North America Albers', nice = TRUE)
```

`customAlbers` *Custom coordinate reference system WKT2 string*

Description

These functions take as input either a spatial object or coordinate pair and a custom WKT2 (well-known text) coordinate reference system string centered on the object or coordinate. Projections include:

- Albers conic equal-area
- Lambert azimuthal equal-area
- Vertical near-side (i.e., as the world appears from geosynchronous orbit)

Please note that these are *NOT* standard projections, so do not have an EPSG or like code.

Usage

```
customAlbers(x)

customLambert(x)

customVNS(x, alt = 35800)
```

Arguments

`x`	Either an object of class `SpatRaster`, `SpatVector`, or `sf`, *or* a numeric vector with two values (longitude and latitude of the center of the projection), *or* a two-column matrix/data frame with the centroid of the projection.
`alt`	Altitude in meters of the viewpoint in km. The default (35800 km) is geosynchronous orbit.

Value

A WKT2 (well-known text) string.

Functions

- `customLambert()`: Custom coordinate reference system WKT2 string
- `customVNS()`: Custom coordinate reference system WKT2 string

See Also

`getCRS`, `customAlbers`, `customLambert`, `customVNS`

Examples

```
library(sf)

### Madagascar
data(mad0)

alb <- customAlbers(mad0)
lamb <- customLambert(mad0)
vert <- customVNS(mad0)

madAlb <- st_transform(mad0, alb)
madLamb <- st_transform(mad0, lamb)
madVert <- st_transform(mad0, vert)

oldPar <- par(mfrow=c(2, 2))

plot(st_geometry(mad0), main='Unprojected (WGS84)')
plot(st_geometry(madAlb), main='Albers')
plot(st_geometry(madLamb), main='Lambert')
plot(st_geometry(madVert), main='Vertical')

par(oldPar)

### Canada
# The effect is more noticeable when plotting large areas,
# especially if they lie near the poles.
# This example can take a few minutes to run and plot.

library(terra)

canFile <- system.file('extdata', 'canada_level0_gadm41.gpkg', package='enmSdmX')
can <- vect(canFile)

alb <- customAlbers(can)
lamb <- customLambert(can)
vert <- customVNS(can)

canAlb <- project(can, alb)
canLamb <- project(can, lamb)
canVert <- project(can, vert)

oldPar <- par(mfrow=c(2, 2))

plot(can, main = 'Unprojected (WGS84)')
plot(canAlb, main = 'Albers')
plot(canLamb, main = 'Lambert')
plot(canVert, main = 'Vertical')

par(oldPar)
```

`decimalToDms` *Convert geographic coordinates in decimal format to degrees-minutes-second*

Description

This function converts geographic coordinates in decimal format to degrees-minutes-seconds (DD-MM-SS) format.

Usage

```
decimalToDms(x)
```

Arguments

x	Numeric or vector of numeric values, longitude or latitude in decimal format.

Value

A numeric matrix with three columns: degrees, seconds, and seconds. Note that the hemisphere (i.e., indicated by the sign of x) is not returned since it could be either north/south or east/west.

Examples

```
decimalToDms(38.56123) # latitude of St. Louis, Missouri, USA
decimalToDms(90.06521) # longitude of St. Louis, Missouri, USA
```

`dmsToDecimal` *Convert geographic coordinates in degrees-minutes-second to decimal format*

Description

This function converts geographic coordinates in degrees-minutes-seconds (DD-MM-SS) format to decimal format.

Usage

```
dmsToDecimal(dd, mm, ss, hemis = NULL)
```

Arguments

dd	Numeric. Degrees longitude or latitude. Can be a decimal value.
mm	Numeric. Minutes longitude or latitude. Can be a decimal value.
ss	Numeric. Second longitude or latitude. Can be a decimal value.
hemis	Character or NULL (default). "N" (north), "S" (south), "E" (east), or "W" (west). If left as NULL, then the value returned will always be positive, even if it is in the western or southern hemisphere.

Value

Numeric.

Examples

```
dmsToDecimal(38, 37, 38) # latitude of St. Louis, Missouri, USA
dmsToDecimal(38, 37, 38, 'N') # latitude of St. Louis, Missouri, USA
dmsToDecimal(90, 11, 52.1) # longitude of St. Louis, Missouri, USA
dmsToDecimal(90, 11, 52.1, 'W') # longitude of St. Louis, Missouri, USA
```

---

elimCellDuplicates *Thin spatial points so that there is but one per raster cell*

Description

This function thins spatial points such that no more than one point falls within each cell of a reference raster. If more than one point falls in a cell, the first point in the input data is retained unless the user specifies a priority for keeping points.

Usage

```
elimCellDuplicates(x, rast, longLat = NULL, priority = NULL)
```

Arguments

x	Points. This can be either a data.frame, matrix, SpatVector, or sf object.
rast	SpatRaster object.
longLat	Two-element character vector *or* two-element integer vector. If x is a data.frame, then this should be a character vector specifying the names of the fields in x *or* a two-element vector of integers that correspond to longitude and latitude (in that order). For example, c('long', 'lat') or c(1, 2). If x is a matrix, then this is a two-element vector indicating the column numbers in x that represent longitude and latitude. For example, c(1, 2). If x is an sf object then this is ignored.

priority — Either NULL, in which case for every cell with more than one point the first point in x is chosen, or a numeric or character vector indicating preference for some points over others when points occur in the same cell. There should be the same number of elements in priority as there are points in x. Priority is assigned by the natural sort order of priority. For example, for 3 points in a cell for which priority is c(2, 1, 3), the script will retain the second point and discard the rest. Similarly, if priority is c('z', 'y', 'x') then the third point will be chosen. Priorities assigned to points in other cells are ignored when thinning points in a particular cell.

Value

Object of class x.

Examples

```
# This example can take >10 second to run.

library(terra)
x <- data.frame(
    long=c(-90.1, -90.1, -90.2, 20),
    lat=c(38, 38, 38, 38), point=letters[1:4]
)
rast <- rast() # empty raster covering entire world with 1-degree resolution
elimCellDuplicates(x, rast, longLat=c(1, 2))
elimCellDuplicates(x, rast, longLat=c(1, 2), priority=c(3, 2, 1, 0))
```

---

evalAUC *Weighted AUC*

Description

This function calculates the area under the receiver-operator characteristic curve (AUC) following Mason & Graham (2002). Each case (presence/non-presence) can be assigned a weight, if desired.

Usage

```
evalAUC(
  pres,
  contrast,
  presWeight = rep(1, length(pres)),
  contrastWeight = rep(1, length(contrast)),
  na.rm = FALSE,
  ...
)
```

Arguments

pres	Vector of predictions for "positive" cases (e.g., predictions at presence sites).
contrast	Vector of predictions for "negative" cases (e.g., predictions at absence/background sites).
presWeight	Weights of positive cases. The default is to assign each positive case a weight of 1.
contrastWeight	Weights of contrast cases. The default is to assign each case a weight of 1.
na.rm	Logical. If TRUE then remove any positive cases and associated weights and contrast predictions and associated weights with NAs.
...	Other arguments (unused).

Value

A Numeric value.

References

Mason, S.J. and N.E. Graham. 2002. Areas beneath the relative operating characteristics (ROC) and relative operating levels (ROL) curves: Statistical significance and interpretation. *Quarterly Journal of the Royal Meteorological Society* 128:2145-2166. doi:10.1256/003590002320603584

See Also

pa_evaluate, evalMultiAUC

Examples

```
pres <- seq(0.5, 1, by=0.1)
contrast <- seq(0, 1, by=0.01)

# unweighted
evalAUC(pres, contrast)

# weighted (weight presences with low predictions more)
presWeight <- c(1, 1, 1, 0.5, 0.5, 0.5)
evalAUC(pres, contrast, presWeight=presWeight)

# weighted (weight presences with high predictions more)
presWeight <- c(0.5, 0.5, 0.5, 1, 1, 1)
evalAUC(pres, contrast, presWeight=presWeight)

# weight presences and absences
contrastWeight <- sqrt(contrast)
evalAUC(pres, contrast, presWeight=presWeight, contrastWeight=contrastWeight)
```

`evalContBoyce` *Continuous Boyce Index (CBI) with weighting*

Description

This function calculates the continuous Boyce index (CBI), a measure of model accuracy for presence-only test data.

Usage

```
evalContBoyce(
  pres,
  contrast,
  numBins = 101,
  binWidth = 0.1,
  presWeight = rep(1, length(pres)),
  contrastWeight = rep(1, length(contrast)),
  autoWindow = TRUE,
  method = "spearman",
  dropZeros = TRUE,
  graph = FALSE,
  table = FALSE,
  na.rm = FALSE,
  ...
)
```

Arguments

`pres`	Numeric vector. Predicted values at presence sites.
`contrast`	Numeric vector. Predicted values at background sites.
`numBins`	Positive integer. Number of (overlapping) bins into which to divide predictions.
`binWidth`	Positive numeric value < 1. Size of a bin. Each bin will be `binWidth * (max - min)`. If `autoWindow` is `FALSE` (the default) then `min` is 0 and `max` is 1. If `autoWindow` is `TRUE` then `min` and `max` are the maximum and minimum value of all predictions in the background and presence sets (i.e., not necessarily 0 and 1).
`presWeight`	Numeric vector same length as `pres`. Relative weights of presence sites. The default is to assign each presence a weight of 1.
`contrastWeight`	Numeric vector same length as `contrast`. Relative weights of background sites. The default is to assign each presence a weight of 1.
`autoWindow`	Logical. If `FALSE` calculate bin boundaries starting at 0 and ending at 1 + epsilon (where epsilon is a very small number to assure inclusion of cases that equal 1 exactly). If `TRUE` (default) then calculate bin boundaries starting at minimum predicted value and ending at maximum predicted value.

method	Character. Type of correlation to calculate. The default is 'spearman', the Spearman rank correlation coefficient used by Boyce et al. (2002) and Hirzel et al. (2006), which is the "traditional" CBI. In contrast, 'pearson' or 'kendall' can be used instead. See cor for more details.
dropZeros	Logical. If TRUE then drop all bins in which the frequency of presences is 0.
graph	Logical. If TRUE then plot P vs E and P/E versus bin.
table	Logical. If TRUE, the function will also return a table of P (proportion of presence weights per bin), E (expected proportion of presence weights per bin–from contrast sites), and the ratio of the two.
na.rm	Logical. If TRUE then remove any presences and associated weights and background predictions and associated weights with NAs.
...	Other arguments (not used).

Details

CBI is the Spearman rank correlation coefficient between the proportion of sites in each prediction class and the expected proportion of predictions in each prediction class based on the proportion of the landscape that is in that class. The index ranges from -1 to 1. Values >0 indicate the model's output is positively correlated with the true probability of presence. Values <0 indicate it is negatively correlated with the true probability of presence.

Value

Numeric value, or if table is TRUE, then a list object with CBI plus a data frame with P (proportion of presence weights per bin), E (expected proportion of presence weights per bin–from contrast sites), and the ratio of the two.

References

Boyce, M.S., Vernier, P.R., Nielsen, S.E., and Schmiegelow, F.K.A. 2002. Evaluating resource selection functions. *Ecological Modeling* 157:281-300. doi:10.1016/S03043800(02)002004

Hirzel, A.H., Le Lay, G., Helfer, V., Randon, C., and Guisan, A. 2006. Evaluating the ability of habitat suitability models to predict species presences. *Ecological Modeling* 199:142-152. doi:10.1016/j.ecolmodel.2006.05.017

See Also

cor, pa_evaluate, evalAUC, evalMultiAUC, evalContBoyce, evalThreshold, evalThresholdStats, evalTjursR2, evalTSS

Examples

```
set.seed(123)
pres <- sqrt(runif(100))
contrast <- runif(1000)
evalContBoyce(pres, contrast)

presWeight <- c(rep(1, 10), rep(0.5, 90))
```

```
evalContBoyce(pres, contrast, presWeight=presWeight)
```

`evalMultiAUC` *Calculate multivariate weighted AUC*

Description

This function calculates a multivariate version of the area under the receiver-operator characteristic curve (AUC). The multivariate version is simply the mean AUC across all possible pairwise AUCs for all cases (Hand & Till 2001). For example, if we have predictions that can be classified into three groups of expectation, say A, B, and C, where we expect predictions assigned to group A are > those in B and C, and predictions in group B are expected to be > those in group C, the multivariate AUC for this situation is `mean(wAB * auc_mean(A, B), wAC * auc_mean(A, C), wBC * auc_mean(B, C))`, where `auc_mean(X, Y)`, is the AUC calculated between cases `X` and `Y`, and `wXY` is a weight for that case-comparison.

Usage

```
evalMultiAUC(..., weightBySize = FALSE, na.rm = FALSE)
```

Arguments

`...`	A set of two or more numeric vectors *or* two or more 2-column matrices or data frames. The objects must be listed in order of *expected* probability. For example, you might have a set of predictions for objects you expect to have a low predicted probability (e.g., long-term absences of an animal), a set that you expect to have middle levels of probability (e.g., sites that were recently vacated), and a set for which you expect a high level of predicted probability (e.g., sites that are currently occupied). In this case you should list the cases in order: low, middle, high. If a 2-column matrix or data frame is supplied, then the first column is assumed to represent predictions and the second assumed to represent site-level weights (see `evalAUC`). Note that site-level weighting is different from case-level weighting.
`weightBySize`	Logical, if `FALSE` (default) then the multivariate measure of AUC will treat all comparisons as equal (e.g., low versus middle will weigh as much as middle versus high), and so will simply be the mean AUC across all possible comparisons. If `TRUE` then multivariate AUC is the weighted mean across all possible comparisons where weights are the number of comparisons between each of the two cases. For example, if a set of "low" predictions ("low") has 10 data points, "middle" has 10, and "high" has 20, then the multivariate AUC will be (10 * low + 10 * middle + 20 * high) / (10 + 10 + 20).
`na.rm`	Logical. If `TRUE` then remove any cases in `...` that are `NA`.

Value

Named numeric vector. The names will appear as `case2_over_case1` (which in this example means the AUC of item #1 in the `...` when compared to the second item in `...`), plus `multivariate` (which is the multivariate AUC).

References

Hand, DJ and Till, RJ. 2001. A simple generalisation of the area under the ROC curve for multiple class classification problems. *Machine Learning* 45:171-186 doi:10.1023/A:1010920819831.

See Also

`pa_evaluate`, `evalAUC`, `evalContBoyce`, `evalThreshold`, `evalThresholdStats`, `evalTjursR2`, `evalTSS`

Examples

```
set.seed(123)

# no weights
low <- runif(10)^2
middle <- runif(10)
high <- sqrt(runif(20))

evalMultiAUC(low, middle, high)

# equal weights
low <- matrix(c(low, rep(1, length(low))), ncol=2)
middle <- matrix(c(middle, rep(1, length(middle))), ncol=2)
high <- matrix(c(high, rep(1, length(high))), ncol=2)
evalMultiAUC(low, middle, high)

# equal weights with weighting by number of comparisons
evalMultiAUC(low, middle, high, weightBySize=TRUE)

# unequal weights
middle[ , 2] <- ifelse(middle[ , 1] > 0.5, 0.1, 1)
evalMultiAUC(low, middle, high)

# unequal weights with weighting by number of comparisons
evalMultiAUC(low, middle, high, weightBySize=TRUE)
```

`evalThreshold` *Weighted thresholds for predictions*

Description

This function is similar to the `threshold` function in the **predicts** package, which calculates thresholds to create binary predictions from continuous values. However, unlike that function, it allows the user to specify weights for presences and absence/background predictions. The output will thus be the threshold that best matches the specified criterion taking into account the relative weights of the input values.

Usage

```
evalThreshold(
  pres,
  contrast,
  presWeight = rep(1, length(pres)),
  contrastWeight = rep(1, length(contrast)),
  at = c("msss", "mdss", "minPres", "prevalence", "sensitivity"),
  sensitivity = 0.9,
  thresholds = seq(0, 1, by = 0.001),
  na.rm = FALSE,
  ...
)
```

Arguments

`pres`	Numeric vector. Predicted values at test presences.
`contrast`	Numeric vector. Predicted values at background/absence sites.
`presWeight`	Numeric vector same length as `pres`. Relative weights of presence sites. The default is to assign each presence a weight of 1.
`contrastWeight`	Numeric vector same length as `contrast`. Relative weights of background sites. The default is to assign each presence a weight of 1.
`at`	Character or character vector, name(s) of threshold(s) to calculate. The default is to calculate them all. • `'msss'`: Threshold that the maximizes the sum of sensitivity and specificity. • `'mdss'`: Threshold that minimizes the difference between sensitivity and specificity. • `'minPres'`: Minimum prediction across presences. This threshold is not weighted. • `'prevalence'`: Prevalence of presences (sum(presence weights) / sum(presence weights + background weights))' • `'sensitivity'`: Threshold that most closely returns the sensitivity specified by `sensitivity`.
`sensitivity`	Value of specificity to match (used only if `at` contains `'sensitivity'`).
`thresholds`	Numeric vector. Thresholds at which to calculate the sum of sensitivity and specificity. The default evaluates all values from 0 to 1 in steps of 0.01.
`na.rm`	Logical. If `TRUE` then remove any presences and associated weights and background predictions and associated weights with `NA`s.
`...`	Other arguments (unused).

Value

Named numeric vector. Fielding, A.H. and J.F. Bell. 1997. A review of methods for the assessment of prediction errors in conservation presence/absence models. *Environmental Conservation* 24:38-49. doi:10.1017/S0376892997000088

See Also

threshold, pa_evaluate, evalAUC, evalMultiAUC, evalContBoyce, evalThresholdStats, evalTjursR2, evalTSS

Examples

```
set.seed(123)

# set of bad and good predictions at presences
bad <- runif(100)^2
good <- runif(100)^0.1
hist(good, breaks=seq(0, 1, by=0.1), border='green', main='Presences')
hist(bad, breaks=seq(0, 1, by=0.1), border='red', add=TRUE)
pres <- c(bad, good)
contrast <- runif(1000)
evalThreshold(pres, contrast)

# upweight bad predictions
presWeight <- c(rep(1, 100), rep(0.1, 100))
evalThreshold(pres, contrast, presWeight=presWeight)

# upweight good predictions
presWeight <- c(rep(0.1, 100), rep(1, 100))
evalThreshold(pres, contrast, presWeight=presWeight)
```

---

evalThresholdStats *Thresholded evaluation statistics*

Description

This function calculates a series of evaluation statistics based on a threshold or thresholds used to convert continuous predictions to binary predictions.

Usage

```
evalThresholdStats(
  thresholds,
  pres,
  contrast,
  presWeight = rep(1, length(pres)),
  contrastWeight = rep(1, length(contrast)),
  delta = 0.001,
```

```
  na.rm = FALSE,
  bg = NULL,
  bgWeight = NULL,
  ...
)
```

Arguments

thresholds	Numeric or numeric vector. Threshold(s) at which to calculate sensitivity and specificity.
pres	Numeric vector. Predicted values at test presences
contrast	Numeric vector. Predicted values at background/absence sites.
presWeight	Numeric vector same length as pres. Relative weights of presence sites. The default is to assign each presence a weight of 1.
contrastWeight	Numeric vector same length as contrast. Relative weights of background sites. The default is to assign each presence a weight of 1.
delta	Positive numeric >0 in the range [0, 1] and usually very small. This value is used only if calculating the SEDI threshold when any true positive rate or false negative rate is 0 or the false negative rate is 1. Since SEDI uses log(x) and log(1 - x), values of 0 and 1 will produce NAs. To obviate this, a small amount can be added to rates that equal 0 and subtracted from rates that equal 1.
na.rm	Logical. If TRUE then remove any presences and associated weights and background predictions and associated weights with NAs.
bg	Same as contrast. Included for backwards compatibility. Ignored if contrast is not NULL.
bgWeight	Same as contrastWeight. Included for backwards compatibility. Ignored if contrastWeight is not NULL.
...	Other arguments (unused).

Value

8-column matrix with the following named columns. a = weight of presences >= threshold, b = weight of backgrounds >= threshold, c = weight of presences < threshold, d = weight of backgrounds < threshold, and N = sum of presence and background weights.

- 'threshold': Threshold
- 'sensitivity': Sensitivity (a / (a + c))
- 'specificity': Specificity (d / (d + b))
- 'ccr': Correct classification rate ((a + d) / N)
- 'ppp': Positive predictive power (a / (a + b))
- 'npp': Negative predictive power (d / (c + d))
- 'mr': Misclassification rate ((b + c) / N)

Fielding, A.H. and J.F. Bell. 1997. A review of methods for the assessment of prediction errors in conservation presence/absence models. *Environmental Conservation* 24:38-49. doi:10.1017/S0376892997000088

See Also

threshold, pa_evaluate, evalAUC, evalMultiAUC, evalContBoyce, evalThreshold, evalTjursR2, evalTSS

Examples

```
set.seed(123)

# set of bad and good predictions at presences
bad <- runif(100)^2
good <- runif(100)^0.1
hist(good, breaks=seq(0, 1, by=0.1), border='green', main='Presences')
hist(bad, breaks=seq(0, 1, by=0.1), border='red', add=TRUE)
pres <- c(bad, good)
contrast <- runif(1000)
thresholds <- c(0.1, 0.5, 0.9)
evalThresholdStats(thresholds, pres, contrast)

# upweight bad predictions
presWeight <- c(rep(1, 100), rep(0.1, 100))
evalThresholdStats(thresholds, pres, contrast, presWeight=presWeight)

# upweight good predictions
presWeight <- c(rep(0.1, 100), rep(1, 100))
evalThresholdStats(thresholds, pres, contrast, presWeight=presWeight)
```

---

evalTjursR2 *Weighted Tjur's R2*

Description

This function calculates Tjur's R2 metric of model discrimination accuracy. Unweighted R2 is simply the difference between the mean predicted value at presence sites and the mean predicted value at absence/background sites. The weighted version allows for differing weights between presences and between absences/contrast values (i.e., the difference between the weighted mean of predictions at presences and weighted mean predictions at absences/contrast locations).

Usage

```
evalTjursR2(
  pres,
  contrast,
  presWeight = rep(1, length(pres)),
  contrastWeight = rep(1, length(contrast)),
  na.rm = FALSE,
  ...
)
```

Arguments

pres	Predictions at presence sites.
contrast	Predictions at absence/background sites.
presWeight	Weights of presence cases. The default is to assign each presence case a weight of 1.
contrastWeight	Weights of absence/background cases. The default is to assign each case a weight of 1.
na.rm	Logical. If TRUE then remove any presences and associated weights and background predictions and associated weights with NAs.
...	Other arguments (unused).

Value

Numeric value.

References

Tjur, T. 2009. Coefficients of determination in logistic regression models-A new proposal: The coefficient of discrimination. *The American Statistician* 63:366-372. doi:10.1198/tast.2009.08210

See Also

pa_evaluate, evalAUC, evalMultiAUC, evalContBoyce, evalThreshold, evalThresholdStats, evalTSS

Examples

```
pres <- seq(0.5, 1, by=0.1)
contrast <- seq(0, 1, by=0.01)

# unweighted
evalTjursR2(pres, contrast)

# weighted (weight presences with low predictions more)
presWeight <- c(1, 1, 1, 0.5, 0.5, 0.5)
evalTjursR2(pres, contrast, presWeight=presWeight)

# weighted (weight presences with high predictions more)
presWeight <- c(0.5, 0.5, 0.5, 1, 1, 1)
evalTjursR2(pres, contrast, presWeight=presWeight)

# weight presences and absences
contrastWeight <- sqrt(contrast)
evalTjursR2(pres, contrast, presWeight=presWeight, contrastWeight=contrastWeight)
```

`evalTSS` *Weighted True Skill Statistic (TSS)*

Description

This function calculates the True Skill Statistic (TSS).

Usage

```
evalTSS(
  pres,
  contrast,
  presWeight = rep(1, length(pres)),
  contrastWeight = rep(1, length(contrast)),
  thresholds = seq(0, 1, by = 0.001),
  na.rm = FALSE,
  ...
)
```

Arguments

`pres`	Numeric vector. Predicted values at test presences
`contrast`	Numeric vector. Predicted values at background/absence sites.
`presWeight`	Numeric vector same length as `pres`. Relative weights of presence sites. The default is to assign each presence a weight of 1.
`contrastWeight`	Numeric vector same length as `contrast`. Relative weights of background sites. The default is to assign each presence a weight of 1.
`thresholds`	Numeric vector. Thresholds at which to calculate the sum of sensitivity and specificity. The default evaluates all values from 0 to 1 in steps of 0.01.
`na.rm`	Logical. If `TRUE` then remove any presences and associated weights and background predictions and associated weights with `NA`s.
`...`	Other arguments (unused).

Details

This function calculates the maximum value of the True Skill Statistic (i.e., across all thresholds, the values that maximizes sensitivity plus specificity).

Value

Numeric value.

References

See Allouche, O., Tsoar, A., and Kadmon, R. 2006. Assessing the accuracy of species distribution models: Prevalence, kappa and the true skill statistic (TSS). *Journal of Applied Ecology* 43:1223-1232. doi:10.1111/j.13652664.2006.01214.x

See Also

pa_evaluate, evalAUC, evalMultiAUC, evalContBoyce, evalThreshold, evalThresholdStats, evalTjursR2

Examples

```
set.seed(123)

# set of bad and good predictions at presences
bad <- runif(30)^2
good <- runif(30)^0.1
hist(good, breaks=seq(0, 1, by=0.1), border='green', main='Presences')
hist(bad, breaks=seq(0, 1, by=0.1), border='red', add=TRUE)
pres <- c(bad, good)
contrast <- runif(1000)
evalTSS(pres, contrast)

# upweight bad predictions
presWeight <- c(rep(1, 30), rep(0.1, 30))
evalTSS(pres, contrast, presWeight=presWeight)

# upweight good predictions
presWeight <- c(rep(0.1, 30), rep(1, 30))
evalTSS(pres, contrast, presWeight=presWeight)
```

---

extentToVect *Convert extent to a spatial polygon*

Description

This function returns a SpatVector or sf polygon representing an extent. The input can be a SpatExtent or sf object, or an object from which a SpatExtent (extent) can be obtained.

Usage

```
extentToVect(x, ...)
```

Arguments

x	A sf, SpatVector, SpatRaster, sf, *or* a vector of four numeric values representing (in this order): x-coordinate of western side of the extent, x-coordinate of eastern side, y-coordinate of the southern side, and y-coordinate of the northern side. If numeric coordinates are supplied, the output will not have a CRS assigned to it unless supplied in ...
...	Arguments to supply to vect.

Value

A `SpatVector` (usual) or, if the input is an `sf` object, an `sf` polygon object.

See Also

plotExtent

Examples

```
data(mad0)
madExtent <- extentToVect(mad0)
plot(madExtent, border='blue', lty='dotted')
plot(mad0[1], add=TRUE)

# NB This is the same as:
library(terra)
madExtent <- ext(mad0)
madExtent <- as.polygons(madExtent, crs = crs(mad0))
```

`geoFold` *Assign geographically-distinct k-folds*

Description

This function generates geographically-distinct cross-validation folds, or "geo-folds" ("g-folds" for short). Points are grouped by proximity to one another. Folds can be forced to have at least a minimum number of points in them. Results are deterministic (i.e., the same every time for the same data).

More specifically, g-folds are created using this process:

- To start, all pairwise distances between points are calculated. These are used in a clustering algorithm to create a dendrogram of relationships by distance. The dendrogram is then "cut" so it has `k` groups (folds). If each fold has at least the minimum desired number of points (`minIn`), then the process stops and fold assignments are returned.
- However, if at least one fold has fewer than the desired number of points, a series of steps is executed.
  - First, the fold with a centroid that is farthest from all others is selected. If it has sufficient points, then the next-most distant fold is selected, and so on.
  - Once a fold is identified that has fewer than the desired number of points, it is grown by adding to it the points closest to its centroid, one at a time. Each time a point is added, the fold centroid is calculated again. The fold is grown until it has the desired number of points. Call this "fold #1". From hereafter, these points are considered "assigned" and not eligible for re-assignment.

- The remaining "unassigned" points are then clustered again, but this time into k - 1 folds. And again, the most-distant group found that has fewer than the desired number of points is found. This fold is then grown as before, using only unassigned points. This fold then becomes "fold #2."
- The process repeats iteratively until there are k folds assigned, each with at least the desired number of points.

The potential downside of this approach is that the last fold is assigned the remainder of points, so will be the largest. One way to avoid gross imbalance is to select the value of minIn such that it divides the points into nearly equally-sized groups.

Usage

```
geoFold(x, k, minIn = 1, longLat = 1:2, method = "complete", ...)
```

Arguments

x	A "spatial points" object of class SpatVector, sf, data.frame, or matrix. If x is a data.frame or matrix, then the points will be assumed to have the WGS84 coordinate system (i.e., unprojected).
k	Numeric: Number of folds to create.
minIn	Numeric: Minimum number of points required to be in a fold.
longLat	Character or integer vector: This is ignored if x is a SpatVector or sf object. However, if x is a data.frame or matrix, then this should be a character or integer vector specifying the columns in x corresponding to longitude and latitude (in that order). For example, c('long', 'lat') or c(1, 2). The default is to assume that the first two columns in x represent coordinates.
method	Character: Method used by hclust to cluster points. By default, this is 'complete', but other methods may give more reasonable results, depending on the case.
...	Additional arguments (unused)

Details

Note that in general it is probably mathematically impossible to cluster points in 2-dimensional space into k groups, each with at least minIn points, in a manner that seems "reasonable" to the eye in all cases. In experimentation, "unreasonable" results often appear when the number of groups is high.

Value

A vector of integers the same length as the number of points in x. Each integer indicates which fold a point in x belongs to.

See Also

geoFoldContrast

Examples

```
library(sf)
library(terra)

# lemur occurrence data
data(mad0)
data(lemurs)
crs <- getCRS('WGS84')
ll <- c('longitude', 'latitude')

# use occurrences of all species... easier to see on map
occs <- st_as_sf(lemurs, coords = ll, crs = getCRS('WGS84'))

# create 100 background points
mad0 <- vect(mad0)
bg <- spatSample(mad0, 100)

### assign 3 folds to occurrences and to background sites
k <- 3
minIn <- floor(nrow(occs) / k) # maximally spread between folds

presFolds <- geoFold(occs, k = k, minIn = minIn)
bgFolds <- geoFoldContrast(bg, pres = occs, presFolds = presFolds)

# number of sites per fold
table(presFolds)
table(bgFolds)

# map
plot(mad0, border = 'gray', main = paste(k, 'geo-folds'))
plot(bg, pch = 3, col = bgFolds + 1, add = TRUE)
plot(st_geometry(occs), pch = 20 + presFolds, bg = presFolds + 1, add = TRUE)

legend(
 'bottomright',
 legend = c(
  'presence fold 1',
  'presence fold 2',
  'presence fold 3',
  'background fold 1',
  'background fold 2',
  'background fold 3'
 ),
 pch = c(21, 22, 23, 3, 3),
 col = c(rep('black', 3), 2, 3),
 pt.bg = c(2, 3, 4, NA, NA)
)
```

`geoFoldContrast` *Assign geographically-distinct k-folds to background/absence sites*

Description

This function generates geographically-distinct cross-validation folds, or "geo-folds" of background or absence sites (i.e., "contrast" sites). Each contrast site is assigned to a fold based on the fold of the presence site that is closest. Typically, this function is run after `geoFold` is run to assign presences to folds.

Usage

```
geoFoldContrast(
  contrast,
  pres,
  presFolds,
  contrastLongLat = 1:2,
  presLongLat = 1:2,
  ...
)
```

Arguments

`contrast`	A "spatial points" object representing contrast sites: • A `SpatVector` or `sf` vector with points • A `data.frame` or `matrix`: Points will be assumed to have the WGS84 coordinate system (i.e., unprojected), and `contrastLongLat` should denote the columns with coordinates.
`pres`	A "spatial points" object representing presence sites: • A `SpatVector` or `sf` vector with points • A `data.frame` or `matrix`: Points will be assumed to have the WGS84 coordinate system (i.e., unprojected), and `presLongLat` should denote the columns with coordinates.
`presFolds`	Numeric vector: These provide the folds to which `pres` are assigned. There must be one value per point in `pres`.
`contrastLongLat, presLongLat`	Character or integer vector: A character or integer vector specifying the columns in `contrast` and `pres` corresponding to longitude and latitude (in that order). The default is to assume that the first two columns in `contrast` represent coordinates. These are ignored if `contrast` or `pres` are a `SpatVector` or an `sf` object.
`...`	Additional arguments (unused)

Value

A vector of integers the same length as the number of points in `contrast`. Each integer indicates which fold a point in `contrast` belongs to.

See Also

`geoFold`

Examples

```
library(sf)
library(terra)

# lemur occurrence data
data(mad0)
data(lemurs)
crs <- getCRS('WGS84')
ll <- c('longitude', 'latitude')

# use occurrences of all species... easier to see on map
occs <- st_as_sf(lemurs, coords = ll, crs = getCRS('WGS84'))

# create 100 background points
mad0 <- vect(mad0)
bg <- spatSample(mad0, 100)

### assign 3 folds to occurrences and to background sites
k <- 3
minIn <- floor(nrow(occs) / k) # maximally spread between folds

presFolds <- geoFold(occs, k = k, minIn = minIn)
bgFolds <- geoFoldContrast(bg, pres = occs, presFolds = presFolds)

# number of sites per fold
table(presFolds)
table(bgFolds)

# map
plot(mad0, border = 'gray', main = paste(k, 'geo-folds'))
plot(bg, pch = 3, col = bgFolds + 1, add = TRUE)
plot(st_geometry(occs), pch = 20 + presFolds, bg = presFolds + 1, add = TRUE)

legend(
 'bottomright',
 legend = c(
  'presence fold 1',
  'presence fold 2',
  'presence fold 3',
  'background fold 1',
  'background fold 2',
  'background fold 3'
 ),
 pch = c(21, 22, 23, 3, 3),
 col = c(rep('black', 3), 2, 3),
 pt.bg = c(2, 3, 4, NA, NA)
)
```

---

geoThin *Thin geographic points deterministically or randomly*

Description

This function thins geographic points such that none have nearest neighbors closer than some user-specified distance. For a given set of points that fall within this distance, thinning can be conducted in two ways. Both begin by first calculating all pairwise distances between points. Then, clusters of points are found based on proximity using the "single-linkage" method (i.e., based on minimum distance between groups). Then, either a deterministic or random method is used to select the retained points:

- Deterministic: For each cluster, distances between each point in the cluster and all points outside of the cluster are calculated. The point retained in each cluster is the one with the greatest minimum pairwise distance to any points in any other cluster. This point will this be maximally isolated from any other point.
- Random: For each cluster, a random point is chosen.

Usage

```
geoThin(x, minDist, random = FALSE, longLat = 1:2, method = "single", ...)
```

Arguments

x	A "spatial points" object of class SpatVector, sf, data.frame, or matrix. If x is a data.frame or matrix, then the points will be assumed to have the WGS84 coordinate system (i.e., unprojected).
minDist	Minimum distance (in meters) needed between points to retain them. Points falling closer than this distance will be candidates for being discarded.
random	Logical: If FALSE (default), then use the deterministic method for thinning. If TRUE, then use the random method.
longLat	Numeric or integer vector: This is ignored if x is a SpatVector or sf object. However, if x is a data.frame or matrix, then this should be a character or integer vector specifying the columns in x corresponding to longitude and latitude (in that order). For example, c('long', 'lat') or c(1, 2). The default is to assume that the first two columns in x represent coordinates.
method	Character: Method used by hclust to cluster points. By default, this is 'single', but in some cases this may result in strange clustering (especially when there is a large number of points). The 'complete' method (or others) may give more reasonable results in these cases.
...	Additional arguments. Not used.

Value

Object of class x.

Examples

```
library(sf)

# lemur occurrence data
data(mad0)
```

```
data(lemurs)
crs <- getCRS('WGS84')
occs <- lemurs[lemurs$species == 'Eulemur fulvus', ]
ll <- c('longitude', 'latitude')
occs <- st_as_sf(occs, coords = ll, crs = getCRS('WGS84'))

# deterministically thin
det <- geoThin(x = occs, minDist = 30000)

# randomly thin
set.seed(123)
rand <- geoThin(x = occs, minDist = 30000, random = TRUE)

# map
oldPar <- par(mfrow = c(1, 2))

plot(st_geometry(occs), cex = 1.4, main = 'Deterministic')
plot(st_geometry(det), pch = 21, cex = 1.4, bg = 1:nrow(det), add = TRUE)
plot(st_geometry(mad0), add = TRUE)

plot(st_geometry(occs), cex = 1.4, main = 'Random')
plot(st_geometry(rand), pch = 21, cex = 1.4, bg = 1:nrow(rand), add = TRUE)
plot(st_geometry(mad0), add = TRUE)

par(oldPar)
```

`getCRS` *WKT string for a named coordinate reference system or a spatial object*

Description

Retrieve the Well-Known text string (WKT2) for a coordinate reference system (CRS) by name or from a spatial object. The most common usage of the function is to return the WKT2 string using an easy-to-remember name. For example, `getCRS('wgs84')` returns the WKT2 string for the WGS84 datum. To get a table of strings, just use `getCRS()`.

Usage

```
getCRS(x = NULL, nice = FALSE, warn = TRUE)
```

Arguments

`x` This can be any of:

- Name of CRS: Each CRS has at least one "alias", which appears in the table returned by `getCRS()`. You can use any of the aliases to refer to a CRS. Within the function, spaces, case, and dashes in aliases are ignored, but to help make the aliases more memorable, the aliases include them.

- NULL (default): This returns a table of projections with their aliases (nearly the same as data(crss)).
- An object of class SpatVector, SpatRaster, or sf. If this is a "Spat" or "G" object, then a character vector with the CRS in WKT form is returned. If a sf is supplied, then a crs object is returned in WKT format.

nice — If TRUE, then print the CRS in a formatted manner and return it invisibly. Default is FALSE.

warn — If TRUE (default), then print a warning if the name of the CRS cannot be found.

Value

A string representing WKT2 (well-known text) object or a data.frame.

Examples

```
# view table of available CRSs
getCRS()

# get specific WKT2 strings
getCRS('WGS84')
getCRS('Mollweide')
getCRS('WorldClim')

# WKT2 strings nice for your eyes
getCRS('WGS84', TRUE)

data(mad0)
getCRS(mad0)
```

---

getValueByCell *Get or assign values to cells in a raster*

Description

These functions get values from a raster at specific cells, or values to specific cells.

Usage

```
getValueByCell(x, cell, format = "raster")

setValueByCell(x, val, cell, format = "raster")
```

Arguments

x	A SpatRaster.
cell	Cell indices. There must be one per value in val.
format	The type of cell indexing used. This can be either "raster" for row indexing (default) or "matrix" for column indexing. Row indexing (the default for rasters), starts with cell "1" in the upper left, cell "2" is to its right, and so on. Numbering then wraps around to the next row. Column indexing (the default for matrices) has the cell "1" in the upper left corner of the matrix. The cell "2" is below it, and so on. The numbering then wraps around to the top of the next column.
val	One or more values. If more the number of cells specified is greater than the number of values in val, then values in val will be recycled.

Value

A data frame (getValueByCell) with cell numbers (in row format), or a SpatRaster (setValueByCell).

See Also

setValues, values

Examples

```
library(terra)
x <- rast(nrow=10, ncol=10)
x[] <- round(10 * runif(100))

cell <- c(1, 20, 40, 80)
getValueByCell(x, cell = cell)
getValueByCell(x, cell = cell, format = 'matrix')

y <- setValueByCell(x, val = 20, cell = cell)
plot(y)
z <- setValueByCell(x, val = 30, cell = cell, format = 'matrix')

plot(c(x, y, z))
```

---

globalx *"Friendly" wrapper for terra::global() for calculating raster statistics*

Description

Calculate "global" statistics across all the values in a raster. This function is a wrapper for global. That function, by default, sets na.rm = FALSE, so any cell that is NA can cause the summary statistic to also be NA (usually undesirable). The function also returns a data.frame, so often needs a further line of code to get the actual value(s). This function sets na.rm = TRUE by default, and returns a numeric vector (not a data.frame).

Usage

```
globalx(x, fun, na.rm = TRUE, ..., weights = NULL)
```

Arguments

x	A SpatRaster.
fun	A function or the name of a function (in quotes). See global for more details.
na.rm	If TRUE (default), then the function in fun will ignore NA cells.
...	Additional arguments to pass to fun.
weights	Either NULL or a SpatRaster.

Value

A numeric vector, one value per layer in x.

Examples

```
library(terra)

r <- rast(ncols=10, nrows=10)
values(r) <- 1:ncell(r)


global(r, 'sum') # terra
globalx(r, 'sum') # enmSdmX

global(r, "mean", na.rm=TRUE)[1, 1] # terra... same as enmSdmX::globalx
```

---

interpolateRasts *Interpolate values from a series of rasters*

Description

This function returns a series of rasters interpolated from another series of rasters. For example, the input might represent rasters of a process measured at times t, t + 1, and t + 4. The rasters at t + 2 and t + 3 could be interpolated based on the values in the other rasters. Note that this function can take a lot of time and memory, even for relatively small rasters.

Usage

```
interpolateRasts(
  rasts,
  interpFrom,
  interpTo,
  type = "linear",
  onFail = NA,
```

```
  useRasts = FALSE,
  na.rm = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

rasts	A "stack" of SpatRasterss.
interpFrom	Numeric vector, one value per raster in rasts. Values represent "distance" along the set of rasters rasters (e.g., time).
interpTo	Numeric vector, values of "distances" at which to interpolate the rasters.
type	Character. The type of model used to do the interpolation. Note that some of these (the first few) are guaranteed to go through every point being interpolated from. The second set, however, are effectively regressions so are not guaranteed to do through *any* of the points. Note that some methods cannot handle cases where at least some series of cells have < a given number of non-NA values (e.g., smooth splines will not work if there are < 4 cells with non-NA values).

- linear: A model based on linear segments "fastened" at each value of interpFrom. The segments will intersect each value being interpolated from.
- spline: A natural splines-based model. Splines will intersect each value being interpolated from.
- gam: A generalized additive model. Note that the GAM is *not* guaranteed to intersect each value being interpolated from. Arguments to gam can be supplied via .... Especially note the family argument! You can use the onFail argument with this method since in some cases gam if there are too few data points.
- glm: A generalized linear model. Note that the GLM is *not* guaranteed to intersect each value being interpolated from. Arguments to gam can be supplied via .... Especially note the family argument (the main reason for why you would use a GLM versus just linear interpolation)! You can use the onFail argument with this method since in some cases glm if there are too few data points.
- ns: A natural splines model. Note that the NS is *not* guaranteed to intersect each value being interpolated from. Arguments to trainNS can be supplied via .... Especially note the family argument and the df argument! If df is not supplied, then the number of splines attempted will be equal to 1:(length(interpFrom) - 1). You can use the onFail argument with this method.
- poly: A polynomial model. This method constructs an n-degree polynomial where n = length(interpFrom) - 1. The most parsimonious model is then selected from all possible subsets of models (including an intercept-only model) using AICc. This method is *not* guaranteed to intersect each value being interpolated from. Arguments to glm can be supplied via .... Especially note the family argument! If family is not supplied, then the response is assumed to have a Gaussian distribution. You can use the onFail argument with this method.

- bs: A basis-spline model. This method constructs a series of models with *n*-degree basis-spline model where *n* ranges from 3 to length(interpFrom) - 1. The most parsimonious model is then selected from all possible subsets of models (including an intercept-only model) using AICc. This method is *not* guaranteed to intersect each value being interpolated from. Arguments to glm can be supplied via .... Especially note the family argument! If family is not supplied, then the response is assumed to have a Gaussian distribution. You can use the onFail argument with this method.
- smooth.spline: A smooth-spline model (see smooth.spline). This method is *not* guaranteed to intersect each value being interpolated from. Arguments to smooth.spline can be supplied via .... Unlike some other methods, a family cannot be specified (Gaussian is assumed)! You can use the onFail argument with this method.

onFail — Either NA (default) or any one of 'linear', 'spline', or 'poly'. If a method specified by type fails (i.e., because there are fewer than the required number of values to interpolate from), this method is used in its place. If this is NA and the method fails, then an error occurs.

useRasts — Logical. If FALSE (default), then the calculations are done using arrays. This can be substantially faster than using rasters (when useRasts = TRUE), but also run into memory issues.

na.rm — Logical, if TRUE (default) then ignore cases where all values in the same cells across rasters from which interpolations are made are NA (i.e., do not throw an error). If FALSE, then throw an error when this occurs.

verbose — Logical. If TRUE (default), display progress.

... — Other arguments passed to approx or spline (*do not* include any of these arguments: x, y, or xout), or to glm, gam, or smooth.spline.

Details

This function can be very memory-intensive for large rasters. It may speed things up (and make them possible) to do interpolations piece by piece (e.g., instead of interpolating between times t0, t1, t2, t3, ..., interpolate between t0 and t1, then t1 and t2, etc. This may give results that differ from using the entire set, however, depending on what type of interpolation is used. Note that using linear and splines will often yield very similar results, except that in a small number of cases splines may produce very extreme interpolated values.

Value

A SpatRaster "stack" with one layer per element in interpTo.

See Also

approximate, approxfun, splinefun, trainNS, glm, , bs, smooth.spline.

Examples

```
# NB: The example below can take a few minutes to run.
```

```
library(terra)

interpFrom <- c(1, 3, 4, 8, 10, 11, 15)
interpTo <- 1:15
rx <- rast(nrows=10, ncols=10)
r1 <- setValues(rx, rnorm(100, 1))
r3 <- setValues(rx, rnorm(100, 3))
r4 <- setValues(rx, rnorm(100, 5))
r8 <- setValues(rx, rnorm(100, 11))
r10 <- setValues(rx, rnorm(100, 3))
r11 <- setValues(rx, rnorm(100, 5))
r15 <- setValues(rx, rnorm(100, 13))
rasts <- c(r1, r3, r4, r8, r10, r11, r15)
names(rasts) <- paste0('rasts', interpFrom)

linear <- interpolateRasts(rasts, interpFrom, interpTo)
spline <- interpolateRasts(rasts, interpFrom, interpTo, type='spline')
gam <- interpolateRasts(rasts, interpFrom, interpTo, type='gam', onFail='linear')
ns <- interpolateRasts(rasts, interpFrom, interpTo, type='ns', onFail='linear', verbose=FALSE)
poly <- interpolateRasts(rasts, interpFrom, interpTo, type='poly', onFail='linear')
bs <- interpolateRasts(rasts, interpFrom, interpTo, type='bs', onFail='linear')
ss <- interpolateRasts(rasts, interpFrom, interpTo, type='smooth.spline', onFail='linear',
verbose=FALSE)

# examine trends for a particular point on the landscape
pts <- matrix(c(-9, 13), ncol = 2)
pts <- vect(pts)
linearExt <- unlist(terra::extract(linear, pts, ID=FALSE))
splineExt <- unlist(terra::extract(spline, pts, ID=FALSE))
gamExt <- unlist(terra::extract(gam, pts, ID=FALSE))
nsExt <- unlist(terra::extract(ns, pts, ID=FALSE))
polyExt <- unlist(terra::extract(poly, pts, ID=FALSE))
bsExt <- unlist(terra::extract(bs, pts, ID=FALSE))
ssExt <- unlist(terra::extract(ss, pts, ID=FALSE))

mins <- min(linearExt, splineExt, gamExt, nsExt, polyExt, bsExt, ssExt)
maxs <- max(linearExt, splineExt, gamExt, nsExt, polyExt, bsExt, ssExt)

plot(interpTo, linearExt, type='l', lwd=2, ylim=c(mins, maxs), ylab='Value')
lines(interpTo, splineExt, col='blue')
lines(interpTo, gamExt, col='green')
lines(interpTo, nsExt, col='orange')
lines(interpTo, polyExt, col='gray')
lines(interpTo, bsExt, col='magenta')
lines(interpTo, ssExt, col='cyan')

ext <- unlist(extract(rasts, pts, ID = FALSE))
points(interpFrom, ext)

legend('topleft', inset=0.01, lty=c(rep(1, 7), NA),
legend=c('linear', 'spline', 'GAM', 'NS', 'polynomial', 'B-spline',
'Smooth spline', 'Observed'), col=c('black', 'blue', 'green',
```

```
'orange', 'gray', 'magenta', 'cyan'), pch=c(rep(NA, 7), 1))
```

`lemurs` *Lemur occurrences from GBIF*

Description

Data frame of lemur occurrences

Usage

```
data(lemurs)
```

Format

An object of class `'data.frame'`.

Source

GBIF

Examples

```
data(lemurs)
lemurs
```

`longLatRasts` *Generate rasters with cell values equal to cell longitude or latitude*

Description

This function generates a raster stack with two rasters, one with cell values equal to the cell's longitude and the other with cell values equal to the cell's latitude.

Usage

```
longLatRasts(x, m = TRUE, filePath = NULL, ...)
```

Arguments

x	SpatRaster object. The output will have the same resolution, extent, and coordinate projection system as x.
m	Any of: • TRUE (default): Calculate longitude and latitude only for cells that are not NA. • FALSE: Calculate longitude and latitude for all cells. • A SpatRaster object: Force any cells that are NA in this raster to also be NA in the output.
filePath	String or NULL. If a string, then this is the path (not including file name) to which to write the raster stack with longitude/latitude rasters. If NULL then no file is written.
...	Arguments to pass to writeRaster (if filePath is not NULL).

Value

Object of class SpatRaster.

Examples

```
library(terra)

# generate long/lat rasters for the world
x <- rast() # raster with 1 deg resolution and extent equal to entire world
x[] <- 1:ncell(x)
longLat <- longLatRasts(x)
plot(longLat)

# demonstrate masking
# randomly force some cells to NA
v <- 1:ncell(x)
n <- 10000
v[sample(v, n)] <- NA
x[] <- v
longLatTRUE <- longLatRasts(x, m = TRUE)
longLatFALSE <- longLatRasts(x, m = FALSE)
rasts <- c(x, longLatTRUE, x, longLatFALSE)
names(rasts) <- c('x', 'long_m_TRUE', 'lat_m_TRUE',
 'x', 'long_m_FALSE', 'lat_m_FALSE')
plot(rasts)
```

---

mad0 *Madagascar spatial object*

Description

Outline of Madagascar from GADM. The geometry has been simplified from the version available in GADM, so pleased do not use this for "official" analyses.

Usage

```
data(mad0, package='enmSdmX')
```

Format

An object of class `sf`.

Source

GADM

Examples

```
library(sf)
data(mad0)
mad0
plot(st_geometry(mad0), main='Madagascar')
```

---

mad1 *Madagascar spatial object*

Description

Outlines of regions ("Faritra") of Madagascar from GADM. The geometry has been simplified from the version available in GADM, so pleased do not use this for "official" analyses.

Usage

```
data(mad1, package='enmSdmX')
```

Format

An object of class `sf`.

Source

GADM

Examples

```
library(sf)
data(mad1)
mad1
plot(st_geometry(mad1), main='Madagascar')
```

`madClim` *Present-day climate rasters for Madagascar*

Description

Rasters representing average climate across 1970-2000 for Madagascar from WorldClim version 2.1. Values of these rasters have been rounded to one digit, so *please do not use these for "official" work*. Please also note that CanESM5 in CMIP6 is known to run "too hot", but is useful here to aid illustration.

Format

An object of class `'SpatRaster'`.

Source

WorldClim

Examples

```
library(terra)
rastFile <- system.file('extdata', 'madClim.tif', package='enmSdmX')
madClim <- rast(rastFile)
plot(madClim)
```

`madClim2030` *Future climate rasters for Madagascar*

Description

Rasters representing average climate across 2021-2040 modeled with CanESM5 for SSP 585 for Madagascar from WorldClim version 2.1. Values of these rasters have been rounded to one digit, so *please do not use these for "official" work*. Please also note that CanESM5 in CMIP6 is known to run "too hot", but is useful here to aid illustration.

Format

An object of class `'SpatRaster'`.

Source

WorldClim

Examples

```
library(terra)
rastFile <- system.file('extdata', 'madClim2030.tif', package='enmSdmX')
madClimFut <- rast(rastFile)
plot(madClimFut)
```

`madClim2050` *Future climate rasters for Madagascar*

Description

Rasters representing average climate across 2041-2060 modeled with CanESM5 for SSP 585 for Madagascar from WorldClim version 2.1. Values of these rasters have been rounded to one digit, so *please do not use these for "official" work*. Please also note that CanESM5 in CMIP6 is known to run "too hot", but is useful here to aid illustration.

Format

An object of class `'SpatRaster'`.

Source

WorldClim

Examples

```
library(terra)
rastFile <- system.file('extdata', 'madClim2050.tif', package='enmSdmX')
madClimFut <- rast(rastFile)
plot(madClimFut)
```

madClim2070 *Future climate rasters for Madagascar*

Description

Rasters representing average climate across 2061-2080 modeled with CanESM5 for SSP 585 for Madagascar from WorldClim version 2.1. Values of these rasters have been rounded to one digit, so *please do not use these for "official" work*. Please also note that CanESM5 in CMIP6 is known to run "too hot", but is useful here to aid illustration.

Format

An object of class `'SpatRaster'`.

Source

WorldClim

Examples

```
library(terra)
rastFile <- system.file('extdata', 'madClim2070.tif', package='enmSdmX')
madClimFut <- rast(rastFile)
plot(madClimFut)
```

madClim2090 *Future climate rasters for Madagascar*

Description

Rasters representing average climate across 2081-2100 modeled with CanESM5 for SSP 585 for Madagascar from WorldClim version 2.1. Values of these rasters have been rounded to one digit, so *please do not use these for "official" work*. Please also note that CanESM5 in CMIP6 is known to run "too hot", but is useful here to aid illustration.

Format

An object of class `'SpatRaster'`.

Source

WorldClim

Examples

```
library(terra)
rastFile <- system.file('extdata', 'madClim2090.tif', package='enmSdmX')
madClimFut <- rast(rastFile)
plot(madClimFut)
```

`modelSize` *Number of response data in a model object*

Description

This function returns the number of response data used in a model (i.e., the sample size). If the data are binary it can return the number of 1s and 0s.

Usage

```
modelSize(x, binary = TRUE, graceful = TRUE)
```

Arguments

`x`	A model object. This can be of many classes, including "gbm", "glm", "gam", "MaxEnt", and so on.
`binary`	If `TRUE` (default) then the number of 1s and 0s in the response data is returned. If `FALSE` then the returned values is the total number of response data.
`graceful`	If `TRUE` (default), then the function returns `NA` if the function cannot determine the sample size from the model object. If `FALSE`, then the function exits with an error.

Value

One or two named integers.

Examples

```
set.seed(123)
y <- runif(1:101)^2
yBinary <- as.integer(y > 0.6)
x <- data.frame(x1=1:101, x2=rnorm(101))
model <- lm(y ~ x1 + x2, data=x)
modelBinary <- glm(yBinary ~ x1 + x2, data=x, family='binomial')
modelSize(model, FALSE)
modelSize(model, TRUE) # not binary input... notice warning
modelSize(modelBinary)
modelSize(modelBinary, FALSE)
```

`nearestEnvPoints` *Extract "most conservative" environments from points and/or polygons*

Description

This function implements the "nearest environmental point" method (Smith et al. 2023) to enable the use of occurrence records geolocated only to a general place (e.g., a country or province), along with occurrences georeferenced with little error. The function returns environments from a set of precisely-geolocated points plus the environment associated with each imprecise record.

Usage

```
nearestEnvPoints(
  rasts,
  pts = NULL,
  polys = NULL,
  centerFrom = "pts",
  pca = TRUE,
  numPcs = 3,
  center = TRUE,
  scale = TRUE,
  rule = "nearest",
  na.rm = TRUE,
  out = "both"
)
```

Arguments

`rasts`	A `SpatRaster` or "stack" of `SpatRasters`. Please also see argument `pca`.
`pts`	A set of spatial points of class `SpatVector` or `sf`.
`polys`	A set of spatial polygons of class `SpatVector` or `sf`.
`centerFrom`	Indicates how to locate the "reference" centroid used to identify single points on each polygon. This is only relevant if both `pts` and `polys` are specified.

- `'pts'`: The default is to use the environmental centroid of `pts`, which finds the centroid of `pts`, then finds the location on the border of each polygon closest to this centroid.
- `'polys'`: This option will first calculate the environmental centroid of each polygon, then the centroid of these points, and then find the location on the border of each polygon closest to this point.
- `'both'`: This option first calculates the environmental centroid of each polygon, then finds the joint centroid of these points plus of `pts`, and lastly locates on the border of each polygon the point closest to this grand centroid.

`pca` If `TRUE` (default) and there is more than one raster specified in `rasts`, then a principal components analysis (PCA) is applied to the values of the rasters before finding the closest points. The returned values are those of the original rasters and the PC scores.

`numPcs` The number of PC axes used to find environmental centroids. This is only used if `pca` is `TRUE`. By default, all axes are used.

`center, scale` Settings for `prcomp`. These indicate if, when calculating the PCA, variables should first be centered and scaled (both `TRUE` by default). If the values in `rasts` are not of the same units, this should almost always be `TRUE`. They are ignored if `pca` is `FALSE`.

`rule` Determines how to identify the single environmental point to associate with each polygon. Options include:

- `'nearest'` (default): Returns the environmental point *closest* to the centroid (i.e., the "nearest environmental point").
- `'farthest'`: Returns the environmental point *farthest* from the centroid (i.e., opposite of the "nearest" point)

`na.rm` If `TRUE` (default), ignore `NA`s when extracting from rasters (e.g., if a point or polygon falls onto an `NA` cell). If `FALSE`, then any `NA`s that overlap a point or polygon will result in an error.

`out` Determines what is returned. Only used if both `pts` and `polys` are provided.

- `'both'` (default): Returns all environmental points. If n is the number of points in `pts` and m the number of polygons in `polys`, then the first `n` rows in the returned data frame refer to the environments of the `pts` and the subsequent m to each `poly`.
- `'pts'`: Returns the environmental values associated with each point.
- `'polys'`: Returns the environmental values on each `poly` polygon closest to the given center.

Details

This function locates a set of points from the environments covered by each polygon using the following procedure, the details of which depend on what arguments are specified:

- Only `pts` is specified: Environments are taken directly from the locations of `pts` in environmental space.
- Only `polys` is specified: Environments are taken from the closest environment of all the environments associated with each each polygon that is closest to the environmental centroid of the environmental centroids of the polygons (that may be confusing, but it is not a typo).
- `pts` and `polys` are specified: Environments are taken from the locations of `pts` plus the environment from each polygon closest to the environmental centroid of `pts`. By default, the function uses the environmental centroid of the precise occurrences in step (1), but this can be changed to the environmental centroid of the centroids of the polygons or the environmental centroid of the points defined by the union of precise occurrence points plus the environmental centroids of the polygons.

The function can alternatively return the points on the vertices of the MCP, or points on the input polygons closest to the reference centroid.

Value

A data frame.

References

Smith, A.B., Murphy, S.J., Henderson, D., and Erickson, K.D. 2023. Including imprecisely georeferenced specimens improves accuracy of species distribution models and estimates of niche breadth. *Global Ecology and Biogeography* In press. Open access pre-print: doi:10.1101/2021.06.10.447988

See Also

nearestGeogPoints for the "nearest geographic point" method, a related approach for geographic space.

Examples

```
# This is a contrived example based on red-bellied lemurs in Madagascar.
# Point locations (which are real data) will be assumed to be "precise"
# records. We will designate a set of Faritas ("counties") to represent
# "imprecise" occurrences that can only be georeferenced to a geopolitical
# unit.

library(sf)
library(terra)

# coordinate reference system
wgs84 <- getCRS('WGS84')

# lemur point data
data(lemurs)
precise <- lemurs[lemurs$species == 'Eulemur rubriventer', ]
ll <- c('longitude', 'latitude')
precise <- sf::st_as_sf(precise[ , ll], coords=ll, crs=wgs84)

# *fake* lemur administrative unit-level data
faritras <- c('Vakinankaratra', 'Haute matsiatra', 'Ihorombe',
'Vatovavy Fitovinany', 'Alaotra-Mangoro', 'Analanjirofo', 'Atsinanana',
'Analamanga', 'Itasy')

data(mad1)
imprecise <- mad1[mad1$NAME_2 %in% faritras, ]

# climate predictors
rastFile <- system.file('extdata/madClim.tif', package='enmSdmX')
rasts <- rast(rastFile)

### Plot environment of points and environments of each polygon closest to
### centroid of environments of points. In this example, we use the first two
### principal component axes to characterize the niche.

# apply Nearest Environmental Point method
envPtsPolys <- nearestEnvPoints(rasts, pts = precise, polys = imprecise,
```

```
 pca = TRUE, numPcs = 2)
envPolys <- nearestEnvPoints(rasts, pts = precise, polys = imprecise, numPcs = 2,
 out = 'polys')
envPts <- nearestEnvPoints(rasts, pts = precise, polys = imprecise, numPcs = 2,
 out = 'pts')

allPolyEnvs <- extract(rasts, imprecise)

# plot occurrences in environmental space
plot(envPtsPolys$PC1, envPtsPolys$PC2, pch=16, col='black',
 xlab='PC1', ylab='PC2')

points(envPolys$PC1, envPolys$PC2, pch=21, bg='orange')

legend(
 'bottomleft',
 inset = 0.01,
 legend = c('precise', 'imprecise (closest)'),
 pch = c(16, 21),
 col = c('black', 'black'),
 pt.bg = c('orange', 'orange')
)

### compare identified environments to all environments across all polygons
################################################################################

env <- as.data.frame(rasts)
pca <- stats::prcomp(env, center=TRUE, scale.=TRUE)

allPolyEnvs <- extract(rasts, imprecise, ID = FALSE)
allPolyEnvsPcs <- predict(pca, allPolyEnvs)
allPolyEnvs <- cbind(allPolyEnvs, allPolyEnvsPcs)

# plot in environmental space
plot(allPolyEnvs$PC1, allPolyEnvs$PC2, pch=16, col='orange',
 xlab='PC1', ylab='PC2')
points(envPts$PC1, envPts$PC2, pch=16)
points(envPolys$PC1, envPolys$PC2, pch=1)
legend(
 'bottomleft',
 inset = 0.01,
 legend = c('precise', 'imprecise (closest)', 'imprecise (all)'),
 pch = c(16, 21, 16),
 col = c('black', 'black', 'orange'),
 pt.bg = c(NA, 'orange')
)

### display niches (minimum convex hulls) estimated
### using just precise or precise + imprecise records
#####################################################

pcs <- c('PC1', 'PC2')
preciseIndices <- chull(envPts[ , pcs])
```

```
preciseImpreciseIndices <- chull(envPtsPolys[ , pcs])

preciseIndices <- c(preciseIndices, preciseIndices[1])
preciseImpreciseIndices <- c(preciseImpreciseIndices,
 preciseImpreciseIndices[1])

preciseOnlyNiche <- envPts[preciseIndices, pcs]
preciseImpreciseNiche <- envPtsPolys[preciseImpreciseIndices, pcs]

# plot in environmental space
plot(allPolyEnvs$PC1, allPolyEnvs$PC2, pch=16, col='orange',
 xlab='PC1', ylab='PC2')
points(envPts$PC1, envPts$PC2, pch=16)
points(envPolys$PC1, envPolys$PC2, pch=1)
lines(preciseImpreciseNiche, col='coral4', lwd=2)
lines(preciseOnlyNiche, lty='dotted')

legend(
 'bottomleft',
 inset = 0.01,
 legend = c('precise', 'imprecise (closest)', 'imprecise (all)',
  'MCP imprecise-only', 'MCP precise + imprecise'),
 pch = c(16, 21, 16, NA, NA),
 col = c('black', 'black', 'orange', 'black', 'coral4'),
 pt.bg = c(NA, 'orange', NA, NA, NA),
 lwd = c(NA, NA, NA, 1, 2),
 lty = c(NA, NA, NA, 'dotted', 'solid')
)
```

`nearestGeogPoints` *Minimum convex polygon from a set of spatial polygons and/or points*

Description

This function implements the "nearest geographic point" method (Smith et al. 2023) to enable the use of occurrence records geolocated only to a general place (e.g., a country or province), along with occurrences georeferenced with little error. The function returns a minimum convex polygon (MCP) constructed from a set of spatial polygons and/or points.

Usage

```
nearestGeogPoints(
  pts = NULL,
  polys = NULL,
  centerFrom = "pts",
  return = "mcp",
  terra = TRUE
)
```

Arguments

pts	Either NULL (default) or a set of spatial points. This can be either a SpatVector (terra package) or POINTS or MULTIPOINTS sf object (sf package). *These must be in an equal-area projection!* This can also be a Spatial object (e.g., SpatialPoints or SpatialPointsDataFrame) from the **sp** package, but *the* sp *package will be deprecated in 2023*.
polys	Either NULL (default), or an object representing spatial polygons of (for example) counties in which a species is known to reside. *This must be in an equal-area projection!*. This object can be either a SpatVector (**terra** package)), or POLYGON, MULTIPOLYGON, LINESTRING, or MULTILINESTRING sf object (**sf** package). This can also be a Spatial object (e.g., SpatialPolygons or SpatialPolygonsDataFrame) from the **sp** package, *the* sp *package will be deprecated in 2023*.
centerFrom	Indicates how to locate the "reference" centroid used to identify points on each polygon. This is only relevant if both pts and polys are not NULL. • 'pts': The default is to use the centroid of pts, which finds the centroid of pts, then finds the location on the border of each polygon closest to this centroid. • 'polys': This option will first calculate the centroid of each polygon, then the centroid of these points, and then find the location on the border of each polygon closest to this point. • 'both': This option first calculates the centroid of each polygon, then finds the joint centroid of these points plus of pts, and lastly locates on the border of each polygon the point closest to this grand centroid.
return	Determines what is returned: • 'mcp' (default): The minimum convex polygon • 'mcpPoints': Points of the vertices of the minimum convex polygon • 'polyPoints': The point on each poly polygon closest to the given center
terra	If TRUE (default), the return an object of class SpatVector. Otherwise, return an object of class sf.

Details

This function constructs a minimum convex polygon (MCP) from a set of spatial points and/or spatial polygons. The manner in which this is done depends on whether polys and/or pts are specified:

- Only pts is supplied: The MCP is constructed directly from the points.
- Only polys is supplied: The MCP is constructed from the point on each polygon closest to the centroid of the centroids of the polygons.
- Both pts and polys are supplied: The MCP is constructed from the combined set of pts *and* from the point on each polygon closest to the centroid of pts. By default, the function uses the centroid of the precise occurrences in step (1), but this can be changed to the centroid of the centroids of the polygons or the centroid of the points defined by the union of precise occurrence points plus the centroids of the polygons.

The function can alternatively return the points on the vertices of the MCP, or points on the input polygons closest to the reference centroid.

Value

`SpatVector`, or `sf` `POLYGON` representing a minimum convex polygon.

References

Smith, A.B., Murphy, S.J., Henderson, D., and Erickson, K.D. 2023. Including imprecisely georeferenced specimens improves accuracy of species distribution models and estimates of niche breadth. *Global Ecology and Biogeography* 32:342-355. doi:10.1111/geb.13628 Open access preprint: doi:10.1101/2021.06.10.447988.

See Also

`nearestEnvPoints` for the "nearest environmental point" method, a related application for estimating niche breadth in environmental space.

Examples

```
library(sf)
library(terra)

#########################################################
### example using SpatVector inputs (terra package) ###
#########################################################

### prepare data
################

# Get coordinate reference systems:
# * WGS84
# * Tananarive (Paris) / Laborde Grid - EPSG:29701
wgs84 <- getCRS('WGS84')
madProj <- getCRS('Madagascar Albers')

# outline of Madagascar faritras
data(mad1)
mad1 <- vect(mad1)
mad1 <- project(mad1, madProj)

# lemur point data
data(lemurs)
redBelly <- lemurs[lemurs$species == 'Eulemur rubriventer', ]
ll <- c('longitude', 'latitude')
redBelly <- vect(redBelly, geom=ll, crs=wgs84)
redBelly <- project(redBelly, madProj)

# *fake* lemur farita-level data
faritras <- c('Toamasina', 'Atsimo-Atsinana',
'Amoron\'i mania', 'Sava', 'Itasy')
```

```
polys <- mad1[mad1$NAME_2 %in% faritras, ]

### apply Nearest Geographic Point method
########################################

# get three kinds of minimum convex polygons (MCPs):

# MCP using just polygons
mcpPolys <- nearestGeogPoints(polys = polys)

# MCP using just points
mcpPts <- nearestGeogPoints(pts = redBelly)

# MCP using points & polys
mcpPolysPoints <- nearestGeogPoints(pts = redBelly, polys = polys)

# compare extent of occurrence (EOO) in m2
expanse(mcpPolys)
expanse(mcpPts)
expanse(mcpPolysPoints)

### plot minimum convex polygons
################################

# MCP from precise occurrences only
plot(mad1, border='gray', main='MCP points only')
plot(polys, col='gray80', add=TRUE)
plot(mcpPts, col=scales::alpha('red', 0.4), add=TRUE)
plot(redBelly, pch=21, bg='red', add=TRUE)

legend('topleft',
legend=c('Precise occurrence', 'Imprecise occurrence', 'MCP'),
fill=c(NA, 'gray', scales::alpha('red', 0.4)),
pch=c(21, NA, NA),
pt.bg=c('red', NA, NA),
border=c(NA, 'black', 'black'))

# MCP from imprecise occurrences only
plot(mad1, border='gray', main='MCP polys only')
plot(polys, col='gray80', add=TRUE)
plot(mcpPolys, col=scales::alpha('orange', 0.4), add=TRUE)
plot(redBelly, pch=21, bg='red', add=TRUE)

legend('topleft',
legend=c('Precise occurrence', 'Imprecise occurrence', 'MCP'),
fill=c(NA, 'gray', scales::alpha('orange', 0.4)),
pch=c(21, NA, NA),
pt.bg=c('red', NA, NA),
border=c(NA, 'black', 'black'))

# MCP from precise and imprecise occurrences
plot(mad1, border='gray', main='MCP polys + points')
plot(polys, col='gray80', add=TRUE)
```

```
plot(mcpPolysPoints, col=scales::alpha('green', 0.4), add=TRUE)
plot(redBelly, pch=21, bg='red', add=TRUE)

legend('topleft',
legend=c('Precise occurrence', 'Imprecise occurrence', 'MCP'),
fill=c(NA, 'gray', scales::alpha('green', 0.4)),
pch=c(21, NA, NA),
pt.bg=c('red', NA, NA),
border=c(NA, 'black', 'black'))

##########################################
### example using sf inputs (sf package) ###
##########################################

### prepare data
################

# Get coordinate reference systems:
# * WGS84
# * Tananarive (Paris) / Laborde Grid - EPSG:29701
madProj <- sf::st_crs(getCRS('Madagascar Albers'))
wgs84 <- getCRS('WGS84')

# outline of Madagascar faritras
data(mad1)
mad1 <- sf::st_transform(mad1, madProj)

# lemur point occurrence data
data(lemurs)
redBelly <- lemurs[lemurs$species == 'Eulemur rubriventer', ]
ll <- c('longitude', 'latitude')
redBelly <- sf::st_as_sf(redBelly[ , ll], crs=wgs84, coords=ll)
redBelly <- sf::st_transform(redBelly, madProj)

# *fake* farita-level occurrences
faritras <- c('Toamasina', 'Atsimo-Atsinana',
'Amoron\'i mania', 'Sava', 'Itasy')
polys <- mad1[mad1$NAME_2 %in% faritras, ]

### apply Nearest Geographic Point method
########################################

# get three kinds of minimum convex polygons (MCPs):

# MCP using just polygons
mcpPolys <- nearestGeogPoints(polys = polys, terra = FALSE)

# MCP using just points
mcpPts <- nearestGeogPoints(pts = redBelly, terra = FALSE)

# MCP using points & polys
mcpPolysPoints <- nearestGeogPoints(pts = redBelly, polys = polys,
terra = FALSE)
```

```
# extent of occurrence (EOO) in m2
sf::st_area(mcpPolys)
sf::st_area(mcpPts)
sf::st_area(mcpPolysPoints)

### plot minimum convex polygons
################################

# MCP from precise occurrences only
plot(st_geometry(mad1), border='gray', main='MCP points only')
plot(st_geometry(polys), col='gray80', add=TRUE)
plot(st_geometry(mcpPts), col=scales::alpha('red', 0.4), add=TRUE)
plot(st_geometry(redBelly), pch=21, bg='red', add=TRUE)

legend('topleft',
legend=c('Precise occurrence', 'Imprecise occurrence', 'MCP'),
fill=c(NA, 'gray', scales::alpha('red', 0.4)),
pch=c(21, NA, NA),
pt.bg=c('red', NA, NA),
border=c(NA, 'black', 'black'))

# MCP from imprecise occurrences only
plot(st_geometry(mad1), border='gray', main='MCP points only')
plot(st_geometry(polys), col='gray80', add=TRUE)
plot(st_geometry(mcpPolys), col=scales::alpha('orange', 0.4), add=TRUE)
plot(st_geometry(redBelly), pch=21, bg='red', add=TRUE)

legend('topleft',
legend=c('Precise occurrence', 'Imprecise occurrence', 'MCP'),
fill=c(NA, 'gray', scales::alpha('orange', 0.4)),
pch=c(21, NA, NA),
pt.bg=c('red', NA, NA),
border=c(NA, 'black', 'black'))

# MCP from precise and imprecise occurrences
plot(st_geometry(mad1), border='gray', main='MCP points only')
plot(st_geometry(polys), col='gray80', add=TRUE)
plot(st_geometry(mcpPolysPoints), col=scales::alpha('green', 0.4), add=TRUE)
plot(st_geometry(redBelly), pch=21, bg='red', add=TRUE)

legend('topleft',
legend=c('Precise occurrence', 'Imprecise occurrence', 'MCP'),
fill=c(NA, 'gray', scales::alpha('green', 0.4)),
pch=c(21, NA, NA),
pt.bg=c('red', NA, NA),
border=c(NA, 'black', 'black'))

### NOTE
# Using SpatVector input (terra package) yields EOOs that are slightly
# larger than using Spatial* (sp) or sf (sf) objects (by about 0.03-0.07%
# in this example). The difference arises because terra::expanse() yields a
# different value than sf::st_area.
```

`nicheOverlapMetrics` *Metrics of niche overlap*

Description

This function calculates several metrics of niche overlap based on predictions for two species (or for the same species but different models) at the same sites.

Usage

```
nicheOverlapMetrics(
  x1,
  x2,
 method = c("meanDiff", "meanAbsDiff", "rmsd", "d", "i", "esp", "cor", "rankCor"),
  w = rep(1, length(x1)),
  na.rm = FALSE,
  ...
)
```

Arguments

`x1`	Numeric. Vector of predictions from a model.
`x2`	Numeric. Vector of predictions from another model.
`method`	Character vector, indicates type of metric to calculate: • `meanDiff`: Average difference • `meanAbsDiff`: Average of absolute values of difference • `rmsd`: Root-mean square deviation • `d`: Schoener's *D* • `i`: Warren's *I* • `esp`: Godsoe's *ESP* • `cor`: Pearson correlation between `x1` and `x2` (will apply `logitAdj()` first unless logit=FALSE). • `rankCor`: Spearman rank correlation.
`w`	Numeric vector. Weights of predictions in `x1` and `x2`.
`na.rm`	Logical. If `TTRUE` then remove elements in `x1` and `2` that are `NA` in *either* `x1` or `x2`.
`...`	Other arguments (not used).

Value

List object with one element per value specified by the argument in `method`.

References

Warren, D.L., Glor, R.E., and Turelli, M. 2008. Environmental niche equivalency versus conservatism: Quantitative approaches to niche evolution. *Evolution* 62:2868-2883. doi:10.1111/j.1558-5646.2008.00482.x

Warren, D.L., Glor, R.E., and Turelli, M. 2008. Erratum. *Evolution* 62:2868-2883. doi:10.1111/j.15585646.2010.01204.x

Godsoe, W. 2014. Inferring the similarity of species distributions using Species' Distribution Models. *Ecography* 37:130-136. doi:10.1111/j.16000587.2013.00403.x

See Also

compareResponse

Examples

```
x1 <- seq(0, 1, length.out=100)
x2 <- x1^2
nicheOverlapMetrics(x1, x2)
```

---

plotExtent *Create spatial polygon same size as a plot*

Description

This function creates a "rectangular" SpatVector object with the same dimensions as a plot window. It is especially useful for cropping subsequent rasters or vector objects to the plot window. A plot must be made before calling this function.

Usage

```
plotExtent(x = NULL)
```

Arguments

x	Either NULL (default), an object of class crs, a coordinate reference string (PROJ6 WKT string), or an object with a coordinate reference system. If any of these is provided, the SpatVector object will have this CRS.

Value

SpatVector

See Also

extentToVect

Examples

```
if (FALSE) {

library(sf)

data(mad0)
plot(st_geometry(mad0))
outline <- plotExtent(mad0)
plot(outline, col='cornflowerblue', lty='dotted')
plot(st_geometry(mad0), add=TRUE)

}
```

`predictEnmSdm` *Generic predict function for SDMs/ENMs*

Description

This is a generic predict function that automatically uses the model common arguments for predicting models of the following types: linear models, generalized linear models (GLMs), generalized additive models (GAMs), random forests, boosted regression trees (BRTs)/gradient boosting machines (GBMs), conditional random forests, MaxEnt, and more.

Usage

```
predictEnmSdm(
  model,
  newdata,
  maxentFun = "terra",
  scale = TRUE,
  cores = 1,
  nrows = nrow(newdata),
  paths = .libPaths(),
  ...
)
```

Arguments

`model`	Object of class `lm`, `glm`, `gam`, `randomForest`, `MaxEnt`, `maxnet`, `prcomp`, `kde`, `gbm`, and possibly others (worth a try!).
`newdata`	Data frame or matrix, or `SpatRaster` with data to which to predict.
`maxentFun`	This argument is only used if the `model` object is a MaxEnt model; otherwise, it is ignored. It takes a value of either `'terra'`, in which case a MaxEnt model is predicted using the default `predict` function from the **terra** package, or `'enmSdmX'` in which case the function `predictMaxEnt` function from the **enmSdmX** package (this package) is used.

scale — Logical. If the model is a GLM trained with trainGLM or trainNS, you can use the scale argument in that function to center and scale the predictors. In the predictEnmSdm function, you can set scale to TRUE to scale the rasters or data frame to which you are training using the centers (means) and scales (standard deviations) used in the mode. Otherwise, it is up to you to ensure variables are properly centered and scaled. This argument only has effect if the model is a GLM trained using trainGLM or trainNS.

cores — Integer >= 1. Number of cores to use when calculating multiple models. Default is 1. This is forced to 1 if newdata is a SpatRaster (i.e., as of now, there is no parallelization when predicting to a raster... sorry!). If you have issues when cores > 1, please see the troubleshooting_parallel_operations guide.

nrows — Number of rows of newdata to predict at a time. This is only used if newdata is a data.frame or matrix. The default is to predict all rows at once, but for very large data frames/matrices this can lead to memory issues in some cases. By setting the number of rows, newdata is divided into chunks, and predictions made to each chunk, which may ease memory limitations. This can be combined with multi-coring (which will increase memory requirements). In this case, all cores combined will get nrows of data. How many rows are too many? You will have to decide depending on the capabilities of your system. For example, predicting the outcome of a GLM on data with 10E6 rows may be fine, but predicting a PCA (with multiple axes) to the data data may require too much memory.

paths — Locations where packages are stored. This is typically not useful to the general user, and is only supplied for when the function is called as a functional.

... — Arguments to pass to the algorithm-specific predict function.

Value

Numeric or SpatRaster.

See Also

predict from the **stats** package, predict from the **terra** package, predictMaxEnt, predictMaxNet

Examples

```
# NB: The examples below show a very basic modeling workflow. They have been
# designed to work fast, not produce accurate, defensible models. They can
# take a few minutes to run.

library(mgcv)
library(sf)
library(terra)
set.seed(123)

### setup data
###############
```

```
# environmental rasters
rastFile <- system.file('extdata/madClim.tif', package='enmSdmX')
madClim <- rast(rastFile)

# coordinate reference system
wgs84 <- getCRS('WGS84')

# lemur occurrence data
data(lemurs)
occs <- lemurs[lemurs$species == 'Eulemur fulvus', ]
occs <- vect(occs, geom=c('longitude', 'latitude'), crs=wgs84)

occs <- elimCellDuplicates(occs, madClim)

occEnv <- extract(madClim, occs, ID = FALSE)
occEnv <- occEnv[complete.cases(occEnv), ]

# create 10000 background sites (or as many as raster can support)
bgEnv <- terra::spatSample(madClim, 20000)
bgEnv <- bgEnv[complete.cases(bgEnv), ]
bgEnv <- bgEnv[1:min(10000, nrow(bgEnv)), ]

# collate occurrences and background sites
presBg <- data.frame(
  presBg = c(
    rep(1, nrow(occEnv)),
    rep(0, nrow(bgEnv))
  )
)

env <- rbind(occEnv, bgEnv)
env <- cbind(presBg, env)

predictors <- c('bio1', 'bio12')

### calibrate models
####################

# Note that all of the trainXYZ functions can made to go faster using the
# "cores" argument (set to just 1, by default). The examples below will not
# go too much faster using more cores because they are simplified, but
# you can try!
cores <- 1

# MaxEnt
mx <- trainMaxEnt(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)
```

```
# MaxNet
mn <- trainMaxNet(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)

# generalized linear model (GLM)
gl <- trainGLM(
 data = env,
 resp = 'presBg',
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 verbose = TRUE,
 cores = cores
)

# generalized additive model (GAM)
ga <- trainGAM(
 data = env,
 resp = 'presBg',
 preds = predictors,
 verbose = TRUE,
 cores = cores
)

# natural splines
ns <- trainNS(
 data = env,
 resp = 'presBg',
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 df = 1:2, # too few values for reliable model(?)
 verbose = TRUE,
 cores = cores
)

# boosted regression trees
envSub <- env[1:1049, ] # subsetting data to run faster
brt <- trainBRT(
 data = envSub,
 resp = 'presBg',
 preds = predictors,
 learningRate = 0.001, # too few values for reliable model(?)
 treeComplexity = c(2, 3), # too few values for reliable model, but fast
 minTrees = 1200, # minimum trees for reliable model(?), but fast
 maxTrees = 1200, # too small for reliable model(?), but fast
 tryBy = 'treeComplexity',
 anyway = TRUE, # return models that did not converge
```

```
 verbose = TRUE,
 cores = cores
)

# random forests
rf <- trainRF(
 data = env,
 resp = 'presBg',
 preds = predictors,
 numTrees = c(100, 500), # using at least 500 recommended, but fast!
 verbose = TRUE,
 cores = cores
)

### make maps of models
#######################

# NB We do not have to scale rasters before predicting GLMs and NSs because we
# used the `scale = TRUE` argument in trainGLM() and trainNS().

mxMap <- predictEnmSdm(mx, madClim)
mnMap <- predictEnmSdm(mn, madClim)
glMap <- predictEnmSdm(gl, madClim)
gaMap <- predictEnmSdm(ga, madClim)
nsMap <- predictEnmSdm(ns, madClim)
brtMap <- predictEnmSdm(brt, madClim)
rfMap <- predictEnmSdm(rf, madClim)

maps <- c(
 mxMap,
 mnMap,
 glMap,
 gaMap,
 nsMap,
 brtMap,
 rfMap
)

names(maps) <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NSs', 'BRTs', 'RFs')
fun <- function() plot(occs, col='black', pch=3, add=TRUE)
plot(maps, fun = fun, nc = 4)

### compare model responses to BIO12 (mean annual precipitation)
################################################################

# make a data frame holding all other variables at mean across occurrences,
# varying only BIO12
occEnvMeans <- colMeans(occEnv, na.rm=TRUE)
occEnvMeans <- rbind(occEnvMeans)
occEnvMeans <- as.data.frame(occEnvMeans)
climFrame <- occEnvMeans[rep(1, 100), ]
rownames(climFrame) <- NULL
```

```
minBio12 <- min(env$bio12)
maxBio12 <- max(env$bio12)
climFrame$bio12 <- seq(minBio12, maxBio12, length.out=100)

predMx <- predictEnmSdm(mx, climFrame)
predMn <- predictEnmSdm(mn, climFrame)
predGl <- predictEnmSdm(gl, climFrame)
predGa <- predictEnmSdm(ga, climFrame)
predNat <- predictEnmSdm(ns, climFrame)
predBrt <- predictEnmSdm(brt, climFrame)
predRf <- predictEnmSdm(rf, climFrame)


plot(climFrame$bio12, predMx,
xlab='BIO12', ylab='Prediction', type='l', ylim=c(0, 1))

lines(climFrame$bio12, predMn, lty='solid', col='red')
lines(climFrame$bio12, predGl, lty='dotted', col='blue')
lines(climFrame$bio12, predGa, lty='dashed', col='green')
lines(climFrame$bio12, predNat, lty=4, col='purple')
lines(climFrame$bio12, predBrt, lty=5, col='orange')
lines(climFrame$bio12, predRf, lty=6, col='cyan')

legend(
   'topleft',
   inset = 0.01,
   legend = c(
 'MaxEnt',
 'MaxNet',
 'GLM',
 'GAM',
 'NS',
 'BRT',
 'RF'
   ),
   lty = c(1, 1:6),
   col = c(
 'black',
 'red',
 'blue',
 'green',
 'purple',
 'orange',
 'cyan'
   ),
   bg = 'white'
)

### compare response curves ###
###############################

modelNames <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NS', 'BRT', 'RF')
```

```
responses <- responseCurves(
 models = list(mx, mn, gl, ga, ns, brt, rf),
 env = bgEnv,
 ref = occEnv,
 vars = predictors,
 modelNames = modelNames
)

print(responses)
```

`predictMaxEnt` *Predict a MaxEnt model object (with optional feature-level permutation)*

Description

Takes a MaxEnt `lambda` object or a MaxEnt object and returns raw or logistic predictions. Its output is the same as the `predict` function from the **terra** package, and in fact, is slower than the function from **terra**. However, this function does allow custom manipulations that those functions do not allow (e.g., permuting product features while leaving other features with the same variables intact). This function does *not* clamp predictions–beyond the range of the training data, it extends the prediction in the direction it was going (up/down/no change). The function is based on Peter D. Wilson's document "Guidelines for computing MaxEnt model output values from a lambdas file". The function has a special feature in that it allows you to permute single variables or combinations of variables in specific features before making predictions. This is potentially useful, for example, if you wanted to determine the relative importance of a quadratic feature for a particular variable in a Maxent model relative to the other features in the model. You can also permute values of a variable regardless of which features they appear in. For product features, you can implement the permutation before or after the values are multiplied together (before often makes for bigger differences in predictions).

Usage

```
predictMaxEnt(
  x,
  data,
  type = "cloglog",
  perm = NULL,
  permLinear = NULL,
  permQuad = NULL,
  permHinge = NULL,
  permThresh = NULL,
  permProd = NULL,
  permProdRule = NULL,
  ...
)
```

Arguments

x	Either a MaxEnt lambda object or a MaxEnt model object
data	Data frame. Data to which to make predictions
type	Character. One of: • 'raw': Maxent "raw" values • 'logistic': Maxent logistic values • 'cloglog' (default): Complementary log-log output
perm	Character vector. Name(s) of variable to permute before calculating predictions. This permutes the variables for *all* features in which they occur. If a variable is named here, it overrides permutation settings for each feature featType. Note that for product features the variable is permuted before the product is taken. This permutation is performed before any subsequent permutations (i.e., so if both variables in a product feature are included in perms, then this is equivalent to using the 'before' rule for permProdRule). Ignored if NULL.
permLinear	Character vector. Names(s) of variables to permute in linear features before calculating predictions. Ignored if NULL.
permQuad	Names(s) of variables to permute in quadratic features before calculating predictions. Ignored if NULL.
permHinge	Character vector. Names(s) of variables to permute in forward/reverse hinge features before calculating predictions. Ignored if NULL.
permThresh	Character vector. Names(s) of variables to permute in threshold features before calculating predictions. Ignored if NULL.
permProd	Character list. A list object of n elements, each of which has two character elements naming the variables to permute if they occur in a product feature. Depending on the value of permProdRule, the function will either permute the individual variables then calculate their product or calculate their product, then permute the product across observations. Any other features containing the variables will produce values as normal. Example: permProd=list(c('precipWinter', 'tempWinter'), c('tempSummer', 'precipFall')). The order of the variables in each element of permProd doesn't matter, so permProd=list(c('temp', 'precip')) is the same as permProd=list(c('precip', 'temp')). Ignored if NULL.
permProdRule	Character. Rule for how permutation of product features is applied: 'before' ==> Permute individual variable values then calculate product; 'after' ==> calculate product then permute across these values. Ignored if permProd is NULL.
...	Extra arguments (not used).

Value

Numeric.

See Also

MaxEnt

Examples

```
# NB: The examples below show a very basic modeling workflow. They have been
# designed to work fast, not produce accurate, defensible models. They can
# take a few minutes to run.

library(mgcv)
library(sf)
library(terra)
set.seed(123)

### setup data
##############

# environmental rasters
rastFile <- system.file('extdata/madClim.tif', package='enmSdmX')
madClim <- rast(rastFile)

# coordinate reference system
wgs84 <- getCRS('WGS84')

# lemur occurrence data
data(lemurs)
occs <- lemurs[lemurs$species == 'Eulemur fulvus', ]
occs <- vect(occs, geom=c('longitude', 'latitude'), crs=wgs84)

occs <- elimCellDuplicates(occs, madClim)

occEnv <- extract(madClim, occs, ID = FALSE)
occEnv <- occEnv[complete.cases(occEnv), ]

# create 10000 background sites (or as many as raster can support)
bgEnv <- terra::spatSample(madClim, 20000)
bgEnv <- bgEnv[complete.cases(bgEnv), ]
bgEnv <- bgEnv[1:min(10000, nrow(bgEnv)), ]

# collate occurrences and background sites
presBg <- data.frame(
  presBg = c(
    rep(1, nrow(occEnv)),
    rep(0, nrow(bgEnv))
  )
)

env <- rbind(occEnv, bgEnv)
env <- cbind(presBg, env)

predictors <- c('bio1', 'bio12')

### calibrate models
####################
```

```
# Note that all of the trainXYZ functions can made to go faster using the
# "cores" argument (set to just 1, by default). The examples below will not
# go too much faster using more cores because they are simplified, but
# you can try!
cores <- 1

# MaxEnt
mx <- trainMaxEnt(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)

# MaxNet
mn <- trainMaxNet(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)

# generalized linear model (GLM)
gl <- trainGLM(
 data = env,
 resp = 'presBg',
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 verbose = TRUE,
 cores = cores
)

# generalized additive model (GAM)
ga <- trainGAM(
 data = env,
 resp = 'presBg',
 preds = predictors,
 verbose = TRUE,
 cores = cores
)

# natural splines
ns <- trainNS(
 data = env,
 resp = 'presBg',
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 df = 1:2, # too few values for reliable model(?)
 verbose = TRUE,
```

```
 cores = cores
)

# boosted regression trees
envSub <- env[1:1049, ] # subsetting data to run faster
brt <- trainBRT(
 data = envSub,
 resp = 'presBg',
 preds = predictors,
 learningRate = 0.001, # too few values for reliable model(?)
 treeComplexity = c(2, 3), # too few values for reliable model, but fast
 minTrees = 1200, # minimum trees for reliable model(?), but fast
 maxTrees = 1200, # too small for reliable model(?), but fast
 tryBy = 'treeComplexity',
 anyway = TRUE, # return models that did not converge
 verbose = TRUE,
 cores = cores
)

# random forests
rf <- trainRF(
 data = env,
 resp = 'presBg',
 preds = predictors,
 numTrees = c(100, 500), # using at least 500 recommended, but fast!
 verbose = TRUE,
 cores = cores
)

### make maps of models
#######################

# NB We do not have to scale rasters before predicting GLMs and NSs because we
# used the `scale = TRUE` argument in trainGLM() and trainNS().

mxMap <- predictEnmSdm(mx, madClim)
mnMap <- predictEnmSdm(mn, madClim)
glMap <- predictEnmSdm(gl, madClim)
gaMap <- predictEnmSdm(ga, madClim)
nsMap <- predictEnmSdm(ns, madClim)
brtMap <- predictEnmSdm(brt, madClim)
rfMap <- predictEnmSdm(rf, madClim)

maps <- c(
 mxMap,
 mnMap,
 glMap,
 gaMap,
 nsMap,
 brtMap,
 rfMap
)
```

```
names(maps) <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NSs', 'BRTs', 'RFs')
fun <- function() plot(occs, col='black', pch=3, add=TRUE)
plot(maps, fun = fun, nc = 4)

### compare model responses to BIO12 (mean annual precipitation)
#####################################################################

# make a data frame holding all other variables at mean across occurrences,
# varying only BIO12
occEnvMeans <- colMeans(occEnv, na.rm=TRUE)
occEnvMeans <- rbind(occEnvMeans)
occEnvMeans <- as.data.frame(occEnvMeans)
climFrame <- occEnvMeans[rep(1, 100), ]
rownames(climFrame) <- NULL

minBio12 <- min(env$bio12)
maxBio12 <- max(env$bio12)
climFrame$bio12 <- seq(minBio12, maxBio12, length.out=100)

predMx <- predictEnmSdm(mx, climFrame)
predMn <- predictEnmSdm(mn, climFrame)
predGl <- predictEnmSdm(gl, climFrame)
predGa <- predictEnmSdm(ga, climFrame)
predNat <- predictEnmSdm(ns, climFrame)
predBrt <- predictEnmSdm(brt, climFrame)
predRf <- predictEnmSdm(rf, climFrame)


plot(climFrame$bio12, predMx,
xlab='BIO12', ylab='Prediction', type='l', ylim=c(0, 1))

lines(climFrame$bio12, predMn, lty='solid', col='red')
lines(climFrame$bio12, predGl, lty='dotted', col='blue')
lines(climFrame$bio12, predGa, lty='dashed', col='green')
lines(climFrame$bio12, predNat, lty=4, col='purple')
lines(climFrame$bio12, predBrt, lty=5, col='orange')
lines(climFrame$bio12, predRf, lty=6, col='cyan')

legend(
   'topleft',
   inset = 0.01,
   legend = c(
 'MaxEnt',
 'MaxNet',
 'GLM',
 'GAM',
 'NS',
 'BRT',
 'RF'
   ),
   lty = c(1, 1:6),
   col = c(
 'black',
```

```
 'red',
 'blue',
 'green',
 'purple',
 'orange',
 'cyan'
   ),
   bg = 'white'
)

### compare response curves ###
###############################

modelNames <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NS', 'BRT', 'RF')

responses <- responseCurves(
 models = list(mx, mn, gl, ga, ns, brt, rf),
 env = bgEnv,
 ref = occEnv,
 vars = predictors,
 modelNames = modelNames
)

print(responses)
```

`predictMaxNet` *Predictions from a MaxNet model*

Description

This function is the same as the `predict` function in the **maxnet** package, except that:

- If the input is a data frame, the output is a vector as output (not a single-column matrix);
- If the input is a `SpatRaster`, the output is a `SpatRaster`;
- The default output is on the cloglog scale;
- The function can be explicitly called (versus doing, say, `maxnet:::predict.maxnet`, which does not work even when that would be really useful...).

Usage

```
predictMaxNet(model, newdata, clamp = TRUE, type = "cloglog", ...)
```

Arguments

model	Object of class maxnet.
newdata	Object of class data.frame or SpatRaster (**terra** package).
clamp	If TRUE (default), predict outside the range of training data by 'clamping' values to the last value.
type	One of: • cloglog (default): Predictions are on a complementary log-log scale. • logistic: Predictions are on a logistic scale (and thus technically the same to several decimal places as predictions from MaxEnt <=3.3.3k, except for differences in default features). • link: Predictions are on the scale of the predictors. • exponential: Predictions are on an exponential ('raw') scale.
...	Other arguments (unused).

Value

Numeric vector or SpatRaster

See Also

predict from the **terra** package, and maxnet (see the predict function therein)

Examples

```
# NB: The examples below show a very basic modeling workflow. They have been
# designed to work fast, not produce accurate, defensible models. They can
# take a few minutes to run.

library(mgcv)
library(sf)
library(terra)
set.seed(123)

### setup data
###############

# environmental rasters
rastFile <- system.file('extdata/madClim.tif', package='enmSdmX')
madClim <- rast(rastFile)

# coordinate reference system
wgs84 <- getCRS('WGS84')

# lemur occurrence data
data(lemurs)
occs <- lemurs[lemurs$species == 'Eulemur fulvus', ]
occs <- vect(occs, geom=c('longitude', 'latitude'), crs=wgs84)
```

```
occs <- elimCellDuplicates(occs, madClim)

occEnv <- extract(madClim, occs, ID = FALSE)
occEnv <- occEnv[complete.cases(occEnv), ]

# create 10000 background sites (or as many as raster can support)
bgEnv <- terra::spatSample(madClim, 20000)
bgEnv <- bgEnv[complete.cases(bgEnv), ]
bgEnv <- bgEnv[1:min(10000, nrow(bgEnv)), ]

# collate occurrences and background sites
presBg <- data.frame(
  presBg = c(
    rep(1, nrow(occEnv)),
    rep(0, nrow(bgEnv))
  )
)

env <- rbind(occEnv, bgEnv)
env <- cbind(presBg, env)

predictors <- c('bio1', 'bio12')

### calibrate models
####################

# Note that all of the trainXYZ functions can made to go faster using the
# "cores" argument (set to just 1, by default). The examples below will not
# go too much faster using more cores because they are simplified, but
# you can try!
cores <- 1

# MaxEnt
mx <- trainMaxEnt(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)

# MaxNet
mn <- trainMaxNet(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)

# generalized linear model (GLM)
```

```
gl <- trainGLM(
 data = env,
 resp = 'presBg',
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 verbose = TRUE,
 cores = cores
)

# generalized additive model (GAM)
ga <- trainGAM(
 data = env,
 resp = 'presBg',
 preds = predictors,
 verbose = TRUE,
 cores = cores
)

# natural splines
ns <- trainNS(
 data = env,
 resp = 'presBg',
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 df = 1:2, # too few values for reliable model(?)
 verbose = TRUE,
 cores = cores
)

# boosted regression trees
envSub <- env[1:1049, ] # subsetting data to run faster
brt <- trainBRT(
 data = envSub,
 resp = 'presBg',
 preds = predictors,
 learningRate = 0.001, # too few values for reliable model(?)
 treeComplexity = c(2, 3), # too few values for reliable model, but fast
 minTrees = 1200, # minimum trees for reliable model(?), but fast
 maxTrees = 1200, # too small for reliable model(?), but fast
 tryBy = 'treeComplexity',
 anyway = TRUE, # return models that did not converge
 verbose = TRUE,
 cores = cores
)

# random forests
rf <- trainRF(
 data = env,
 resp = 'presBg',
 preds = predictors,
 numTrees = c(100, 500), # using at least 500 recommended, but fast!
 verbose = TRUE,
 cores = cores
```

```
)

### make maps of models
#######################

# NB We do not have to scale rasters before predicting GLMs and NSs because we
# used the `scale = TRUE` argument in trainGLM() and trainNS().

mxMap <- predictEnmSdm(mx, madClim)
mnMap <- predictEnmSdm(mn, madClim)
glMap <- predictEnmSdm(gl, madClim)
gaMap <- predictEnmSdm(ga, madClim)
nsMap <- predictEnmSdm(ns, madClim)
brtMap <- predictEnmSdm(brt, madClim)
rfMap <- predictEnmSdm(rf, madClim)

maps <- c(
 mxMap,
 mnMap,
 glMap,
 gaMap,
 nsMap,
 brtMap,
 rfMap
)

names(maps) <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NSs', 'BRTs', 'RFs')
fun <- function() plot(occs, col='black', pch=3, add=TRUE)
plot(maps, fun = fun, nc = 4)

### compare model responses to BIO12 (mean annual precipitation)
################################################################

# make a data frame holding all other variables at mean across occurrences,
# varying only BIO12
occEnvMeans <- colMeans(occEnv, na.rm=TRUE)
occEnvMeans <- rbind(occEnvMeans)
occEnvMeans <- as.data.frame(occEnvMeans)
climFrame <- occEnvMeans[rep(1, 100), ]
rownames(climFrame) <- NULL

minBio12 <- min(env$bio12)
maxBio12 <- max(env$bio12)
climFrame$bio12 <- seq(minBio12, maxBio12, length.out=100)

predMx <- predictEnmSdm(mx, climFrame)
predMn <- predictEnmSdm(mn, climFrame)
predGl <- predictEnmSdm(gl, climFrame)
predGa <- predictEnmSdm(ga, climFrame)
predNat <- predictEnmSdm(ns, climFrame)
predBrt <- predictEnmSdm(brt, climFrame)
predRf <- predictEnmSdm(rf, climFrame)
```

```
plot(climFrame$bio12, predMx,
xlab='BIO12', ylab='Prediction', type='l', ylim=c(0, 1))

lines(climFrame$bio12, predMn, lty='solid', col='red')
lines(climFrame$bio12, predGl, lty='dotted', col='blue')
lines(climFrame$bio12, predGa, lty='dashed', col='green')
lines(climFrame$bio12, predNat, lty=4, col='purple')
lines(climFrame$bio12, predBrt, lty=5, col='orange')
lines(climFrame$bio12, predRf, lty=6, col='cyan')

legend(
   'topleft',
   inset = 0.01,
   legend = c(
 'MaxEnt',
 'MaxNet',
 'GLM',
 'GAM',
 'NS',
 'BRT',
 'RF'
   ),
   lty = c(1, 1:6),
   col = c(
 'black',
 'red',
 'blue',
 'green',
 'purple',
 'orange',
 'cyan'
   ),
   bg = 'white'
)

### compare response curves ###
###############################

modelNames <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NS', 'BRT', 'RF')

responses <- responseCurves(
 models = list(mx, mn, gl, ga, ns, brt, rf),
 env = bgEnv,
 ref = occEnv,
 vars = predictors,
 modelNames = modelNames
)

print(responses)
```

`responseCurves` *Plot response curves for one or more models*

Description

This function creates plots of response curves for one or more models. Response curves show how model predictions change as a particular predictor is changed, while holding all other predictors constant. The function can plot response curves for a single model or for multiple models, and can plot response curves for multiple predictors across multiple models.

Usage

```
responseCurves(
  models,
  env,
  ref = NULL,
  vars = NULL,
  modelNames = NULL,
  constantFx = mean,
  combine = TRUE,
  ncol = NULL,
  n = 200
)
```

Arguments

`models`	Either a model object (like `glm`, `gam`, etc. – any model produced by a `trainXYZ()` function), or a `list` of such model objects. If a single model object is passed, the function will create response curves for that model. If a list of model objects is passed, the function will create response curves for each model in the list. The models must be able to be passed to `predictEnmSdm`.
`env`	A `data.frame` containing environmental data. Typically, this data represents the range of environmental conditions over which predictions will be made. The data must contain one column for each predictor variable in the model(s) for which response curves will be plotted. Values do not need to be in a particular order.
`ref`	Either `NULL` (default), a `data.frame`, or a named vector of number ic values. These are used for setting the value of non-focal variables. • `NULL`: The function will use the mean value of each predictor in `env` as the "constant" value for each predictor. • `data.frame`: Mean values across each predictor are used as their "constant" values. • `named vector`: The names of the vector should correspond to the names of the predictors in `env`. The values of the vector should be the "constant" values for each predictor.

vars	Either NULL (default) or a character vector of predictor names. If NULL, response curves will be plotted for all predictors in env. If a character vector, response curves will be plotted for the predictors specified in vars.
modelNames	Either NULL or a character vector of model names. If NULL (default), the names of the models will be set to 'Model 1', 'Model 2', etc. If a character vector, the names of the models will be set to the values in modelNames. The length of modelNames must be equal to the number of models passed to models.
constantFx	Function used to calculate the constant value for each predictor. The default is mean. The function must take a numeric vector as input and return a single numeric value. The function is applied to each column of ref to "calculate" the constant value for each predictor.
combine	Logical. If TRUE (default), graphs will be combined using plot_grid. If FALSE, the output will be a list object, with one plot per element.
ncol	Number of columns to use when combining plots. If combine = FALSE, this argument is ignored. Default is NULL, in which case the function will use the square root of the number of plots as the number of columns.
n	Number of points to use for each variable in the response curve. Default is 200.

Value

Either a **ggplot** grob object or a list of **ggplot** grob objects.

Examples

```
# NB: The examples below show a very basic modeling workflow. They have been
# designed to work fast, not produce accurate, defensible models. They can
# take a few minutes to run.

library(mgcv)
library(sf)
library(terra)
set.seed(123)

### setup data
##############

# environmental rasters
rastFile <- system.file('extdata/madClim.tif', package='enmSdmX')
madClim <- rast(rastFile)

# coordinate reference system
wgs84 <- getCRS('WGS84')

# lemur occurrence data
data(lemurs)
occs <- lemurs[lemurs$species == 'Eulemur fulvus', ]
occs <- vect(occs, geom=c('longitude', 'latitude'), crs=wgs84)

occs <- elimCellDuplicates(occs, madClim)
```

```
occEnv <- extract(madClim, occs, ID = FALSE)
occEnv <- occEnv[complete.cases(occEnv), ]

# create 10000 background sites (or as many as raster can support)
bgEnv <- terra::spatSample(madClim, 20000)
bgEnv <- bgEnv[complete.cases(bgEnv), ]
bgEnv <- bgEnv[1:min(10000, nrow(bgEnv)), ]

# collate occurrences and background sites
presBg <- data.frame(
  presBg = c(
    rep(1, nrow(occEnv)),
    rep(0, nrow(bgEnv))
  )
)

env <- rbind(occEnv, bgEnv)
env <- cbind(presBg, env)

predictors <- c('bio1', 'bio12')

### calibrate models
####################

# Note that all of the trainXYZ functions can made to go faster using the
# "cores" argument (set to just 1, by default). The examples below will not
# go too much faster using more cores because they are simplified, but
# you can try!
cores <- 1

# MaxEnt
mx <- trainMaxEnt(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)

# MaxNet
mn <- trainMaxNet(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)

# generalized linear model (GLM)
gl <- trainGLM(
```

```
 data = env,
 resp = 'presBg',
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 verbose = TRUE,
 cores = cores
)

# generalized additive model (GAM)
ga <- trainGAM(
 data = env,
 resp = 'presBg',
 preds = predictors,
 verbose = TRUE,
 cores = cores
)

# natural splines
ns <- trainNS(
 data = env,
 resp = 'presBg',
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 df = 1:2, # too few values for reliable model(?)
 verbose = TRUE,
 cores = cores
)

# boosted regression trees
envSub <- env[1:1049, ] # subsetting data to run faster
brt <- trainBRT(
 data = envSub,
 resp = 'presBg',
 preds = predictors,
 learningRate = 0.001, # too few values for reliable model(?)
 treeComplexity = c(2, 3), # too few values for reliable model, but fast
 minTrees = 1200, # minimum trees for reliable model(?), but fast
 maxTrees = 1200, # too small for reliable model(?), but fast
 tryBy = 'treeComplexity',
 anyway = TRUE, # return models that did not converge
 verbose = TRUE,
 cores = cores
)

# random forests
rf <- trainRF(
 data = env,
 resp = 'presBg',
 preds = predictors,
 numTrees = c(100, 500), # using at least 500 recommended, but fast!
 verbose = TRUE,
 cores = cores
)
```

```
### make maps of models
#######################

# NB We do not have to scale rasters before predicting GLMs and NSs because we
# used the `scale = TRUE` argument in trainGLM() and trainNS().

mxMap <- predictEnmSdm(mx, madClim)
mnMap <- predictEnmSdm(mn, madClim)
glMap <- predictEnmSdm(gl, madClim)
gaMap <- predictEnmSdm(ga, madClim)
nsMap <- predictEnmSdm(ns, madClim)
brtMap <- predictEnmSdm(brt, madClim)
rfMap <- predictEnmSdm(rf, madClim)

maps <- c(
 mxMap,
 mnMap,
 glMap,
 gaMap,
 nsMap,
 brtMap,
 rfMap
)

names(maps) <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NSs', 'BRTs', 'RFs')
fun <- function() plot(occs, col='black', pch=3, add=TRUE)
plot(maps, fun = fun, nc = 4)

### compare model responses to BIO12 (mean annual precipitation)
################################################################

# make a data frame holding all other variables at mean across occurrences,
# varying only BIO12
occEnvMeans <- colMeans(occEnv, na.rm=TRUE)
occEnvMeans <- rbind(occEnvMeans)
occEnvMeans <- as.data.frame(occEnvMeans)
climFrame <- occEnvMeans[rep(1, 100), ]
rownames(climFrame) <- NULL

minBio12 <- min(env$bio12)
maxBio12 <- max(env$bio12)
climFrame$bio12 <- seq(minBio12, maxBio12, length.out=100)

predMx <- predictEnmSdm(mx, climFrame)
predMn <- predictEnmSdm(mn, climFrame)
predGl <- predictEnmSdm(gl, climFrame)
predGa <- predictEnmSdm(ga, climFrame)
predNat <- predictEnmSdm(ns, climFrame)
predBrt <- predictEnmSdm(brt, climFrame)
predRf <- predictEnmSdm(rf, climFrame)
```

```
plot(climFrame$bio12, predMx,
xlab='BIO12', ylab='Prediction', type='l', ylim=c(0, 1))

lines(climFrame$bio12, predMn, lty='solid', col='red')
lines(climFrame$bio12, predGl, lty='dotted', col='blue')
lines(climFrame$bio12, predGa, lty='dashed', col='green')
lines(climFrame$bio12, predNat, lty=4, col='purple')
lines(climFrame$bio12, predBrt, lty=5, col='orange')
lines(climFrame$bio12, predRf, lty=6, col='cyan')

legend(
   'topleft',
   inset = 0.01,
   legend = c(
 'MaxEnt',
 'MaxNet',
 'GLM',
 'GAM',
 'NS',
 'BRT',
 'RF'
   ),
   lty = c(1, 1:6),
   col = c(
 'black',
 'red',
 'blue',
 'green',
 'purple',
 'orange',
 'cyan'
   ),
   bg = 'white'
)

### compare response curves ###
###############################

modelNames <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NS', 'BRT', 'RF')

responses <- responseCurves(
 models = list(mx, mn, gl, ga, ns, brt, rf),
 env = bgEnv,
 ref = occEnv,
 vars = predictors,
 modelNames = modelNames
)

print(responses)
```

sampleRast *Sample random points from a raster with/out replacement*

Description

This function returns coordinates randomly located on a raster where cells can be sampled with replacement (if desired) and where the probability of selection is proportionate to the cell value, cell area, or the product of cell value times cell area.

Usage

```
sampleRast(x, n, adjArea = TRUE, replace = TRUE, prob = TRUE)
```

Arguments

x	SpatRaster object.
n	Positive integer. Number of points to draw.
adjArea	If TRUE (default) then adjust probabilities so sampling accounts for cell area.
replace	If TRUE (default) then sample with replacement.
prob	If TRUE (default) then sample cells with probabilities proportional to cell values. If adjArea is also TRUE then probabilities are drawn proportional to the product of cell area * the value of the cell.

Value

2-column matrix with longitude and latitude of random points.

See Also

spatSample

Examples

```
library(terra)
r <- rast()
nc <- ncell(r)
r[] <- 1:nc
rands1 <- sampleRast(r, 10000)
rands2 <- sampleRast(r, 10000, adjArea=FALSE)
rands3 <- sampleRast(r, 10000, prob=FALSE)
rands4 <- sampleRast(r, 10000, adjArea=FALSE, prob=FALSE)

oldPar <- par(mfrow=c(2, 2))

plot(r, main='adjArea = TRUE & prob = TRUE')
points(rands1, pch='.')
plot(r, main='adjArea = FALSE & prob = TRUE')
points(rands2, pch='.')
```

```
plot(r, main='adjArea = TRUE & prob = FALSE')
points(rands3, pch='.')
plot(r, main='adjArea = FALSE & prob = FALSE')
points(rands4, pch='.')

par(oldPar)
```

---

spatVectorToSpatial *Convert SpatVector to Spatial**

Description

This function converts a SpatVector object from the **terra** package to a Spatial object of the appropriate class (SpatialPoints, SpatialPointsDataFrame, SpatialPolygons, or SpatialPolygonsDataFrame) from the **sp** package. Note that **sp** is to be retired in 2023, so this function is to be come useful only for legacy applications.

Usage

```
spatVectorToSpatial(x)
```

Arguments

x	SpatVector object.

Value

Object of class Spatial.

Examples

```
library(terra)
f <- system.file('ex/lux.shp', package='terra')
v <- vect(f)
spat <- spatVectorToSpatial(v)
class(spat)
```

squareCellRast *Create a raster with square cells*

Description

This function creates a raster from an object with an extent (i.e., another raster or similar spatial object) with square cells. The user can specify cell resolution (linear dimension) *or* the approximate number of cells desired.

Usage

```
squareCellRast(x, numCells = NULL, res = NULL, vals = NULL)
```

Arguments

x	An object with a spatial extent property (e.g., a SpatRaster or a SpatVector).
numCells	Positive integer, approximate number of cells desired. If this is specified, then res is ignored. If this number of cells cannot be fit into the desired extent exactly, then the actual number of cells will be larger.
res	Positive numeric. Size of a cell in the units of the projection of x (typically meters). Ignored if numCells is not NULL.
vals	Numeric, value to assign to cells. Note that if this is shorter than the number of cells in the output, then values will be recycled. If longer, then values will be truncated. The default is to assign all 0s.

Value

SpatRaster object. The raster will have an extent of the same size or larger than the extent of x.

Examples

```
library(sf)
library(terra)

# project outline of Madagascar to equal-area:
data(mad0)
mad0Ea <- st_transform(mad0, getCRS('madAlbers'))

n <- 101
cellSize_meters <- 10E4
byNumCells <- squareCellRast(mad0Ea, numCells=n)
byCellSize <- squareCellRast(mad0Ea, res=cellSize_meters)

oldPar <- par(mfrow=c(1, 2))

main1 <- paste0('Cells: ', n, ' desired, ', ncell(byNumCells), ' actual')
plot(byNumCells, main = main1)
plot(mad0Ea, add = TRUE)
```

```
main2 <- paste0('Cells ', cellSize_meters, ' m on a side')
plot(byCellSize, main = main2)
plot(mad0Ea, add = TRUE)

par(oldPar)

# Note that in this example they look the same, but the one on the left
# has one less row than the one on the right.
```

---

summaryByCrossValid *Summarize distribution/niche model cross-validation object*

Description

This function summarizes models calibrated using the trainByCrossValid function. It returns aspects of the best models across k-folds (the particular aspects depends on the kind of models used).

Usage

```
summaryByCrossValid(
  x,
  metric = "cbiTest",
  decreasing = TRUE,
  interceptOnly = TRUE
)
```

Arguments

x	The output from the trainByCrossValid function (which is a list). Note that the object *must* include a sublist named tuning.
metric	Metric by which to select the best model in each k-fold. This can be any of the columns that appear in the data frames in x$tuning (or any columns added manually), but typically is one of the following *plus* either Train, Test, or Delta (e.g., 'logLossTrain', 'logLossTest', or 'logLossDelta'): • 'logLoss': Log loss. • 'cbi': Continuous Boyce Index (CBI). Calculated with evalContBoyce. • 'auc': Area under the receiver-operator characteristic curve (AUC). Calculated with evalAUC. • 'tss': Maximum value of the True Skill Statistic. Calculated with evalTSS. • 'msss': Sensitivity and specificity calculated at the threshold that maximizes sensitivity (true presence prediction rate) plus specificity (true absence prediction rate). • 'mdss': Sensitivity (se) and specificity (sp) calculated at the threshold that minimizes the difference between sensitivity and specificity.

- 'minTrainPres': Sensitivity and specificity at the greatest threshold at which all training presences are classified as "present".
- 'trainSe95' and/or 'trainSe90': Sensitivity at the threshold that ensures either 95

decreasing Logical, if TRUE (default), for each k-fold sort models by the value listed in metric in decreasing order (highest connotes "best", lowest "worst"). If FALSE use the lowest value of metric.

interceptOnly Logical. If TRUE (default) and the top models in each case were intercept-only models, return an emppty data frame (with a warning). If FALSE, return results using the first model in each fold that was not an intercept-only model. This is only used if the training function was a generalized linear model (GLM), natural splines model (NS), or generalized additive model (GAM).

Value

Data frame with statistics on the best set of models across k-folds. In each case, the "best" model for each fold is used. Depending on the model algorithm, this output is a data frame with these fields:

- BRTs (boosted regression trees): Learning rate, tree complexity, and bag fraction.
- GLMs (generalized linear models): Frequency of use of each term in the best models.
- MaxEnt and MaxNet: Frequency with which each combination of feature classes was used and mean master regularization multiplier.
- NSs (natural splines): One row per fold and one column per predictor, with values representing the maximum degrees of freedom used for each variable in the best model of each fold.
- RFs (random forests): One row per fold, with values representing the optimal value of numTrees and mtry (see ranger).

See Also

trainByCrossValid, trainBRT, trainGAM, trainGLM, trainMaxEnt, trainNS, trainRF

Examples

```
# The example below show a very basic modeling workflow. It has been
# designed to work fast, not produce accurate, defensible models.
# The general idea is to calibrate a series of models and evaluate them
# against a withheld set of data. One can then use the series of models
# of the top models to better select a "final" model.

## Not run:
# Running the entire set of commands can take a few minutes. This can
# be sped up by increasing the number of cores used. The examples below use
# one core, but you can change that argument according to your machine's
# capabilities.

library(sf)
library(terra)
set.seed(123)
```

```
### setup data
##############

# environmental rasters
rastFile <- system.file('extdata/madClim.tif', package='enmSdmX')
madClim <- rast(rastFile)

# coordinate reference system
wgs84 <- getCRS('WGS84')

# lemur occurrence data
data(lemurs)
occs <- lemurs[lemurs$species == 'Eulemur fulvus', ]
occs <- vect(occs, geom=c('longitude', 'latitude'), crs=wgs84)

occs <- elimCellDuplicates(occs, madClim)

occEnv <- extract(madClim, occs, ID = FALSE)
occEnv <- occEnv[complete.cases(occEnv), ]

# create background sites (using just 1000 to speed things up!)
bgEnv <- terra::spatSample(madClim, 3000)
bgEnv <- bgEnv[complete.cases(bgEnv), ]
bgEnv <- bgEnv[sample(nrow(bgEnv), 1000), ]

# collate occurrences and background sites
presBg <- data.frame(
   presBg = c(
      rep(1, nrow(occEnv)),
      rep(0, nrow(bgEnv))
   )
)

env <- rbind(occEnv, bgEnv)
env <- cbind(presBg, env)

predictors <- c('bio1', 'bio12')

# using "vector" form of "folds" argument
folds <- predicts::kfold(env, 3) # just 3 folds (for speed)

### calibrate models
####################

cores <- 1 # increase this to go faster, if your computer handles it

## MaxEnt
mxx <- trainByCrossValid(
 data = env,
 resp = 'presBg',
 preds = c('bio1', 'bio12'),
 folds = folds,
```

```
 trainFx = trainMaxEnt,
 regMult = 1:2, # too few values for valid model, but fast!
 verbose = 1,
 cores = cores
)

# summarize MaxEnt feature sets and regularization across folds
summaryByCrossValid(mxx)

## MaxNet
mnx <- trainByCrossValid(
 data = env,
 resp = 'presBg',
 preds = c('bio1', 'bio12'),
 folds = folds,
 trainFx = trainMaxNet,
 regMult = 1:2, # too few values for valid model, but fast!
 verbose = 1,
 cores = cores
)

# summarize MaxEnt feature sets and regularization across folds
summaryByCrossValid(mnx)

## generalized linear models
glx <- trainByCrossValid(
 data = env,
 resp = 'presBg',
 preds = c('bio1', 'bio12'),
 folds = folds,
 trainFx = trainGLM,
 verbose = 1,
 cores = cores
)

# summarize GLM terms in best models
summaryByCrossValid(glx)

## generalized additive models
gax <- trainByCrossValid(
 data = env,
 resp = 'presBg',
 preds = c('bio1', 'bio12'),
 folds = folds,
 trainFx = trainGAM,
 verbose = 1,
 cores = cores
)

# summarize GAM terms in best models
summaryByCrossValid(gax)

## natural splines
```

```
nsx <- trainByCrossValid(
 data = env,
 resp = 'presBg',
 preds = c('bio1', 'bio12'),
 folds = folds,
 trainFx = trainNS,
 df = 1:2,
 verbose = 1,
 cores = cores
)

# summarize NS terms in best models
summaryByCrossValid(nsx)

## boosted regression trees
brtx <- trainByCrossValid(
 data = env,
 resp = 'presBg',
 preds = c('bio1', 'bio12'),
 folds = folds,
 trainFx = trainBRT,
 learningRate = c(0.001, 0.0001), # too few values for reliable model(?)
 treeComplexity = c(2, 4), # too few values for reliable model, but fast
 minTrees = 1000,
 maxTrees = 1500, # too small for reliable model(?), but fast
 tryBy = 'treeComplexity',
 anyway = TRUE, # return models that did not converge
 verbose = 1,
 cores = cores
)

# summarize BRT parameters across best models
summaryByCrossValid(brtx)

## random forests
rfx <- trainByCrossValid(
 data = env,
 resp = 'presBg',
 preds = c('bio1', 'bio12'),
 folds = folds,
 trainFx = trainRF,
 verbose = 1,
 cores = cores
)

# summarize RF parameters in best models
summaryByCrossValid(rfx)


## End(Not run)
```

trainBRT *Calibrate a boosted regression tree (generalized boosting machine) model*

Description

This function calibrates a boosted regression tree (or gradient boosting machine) model, and is a wrapper for gbm. The function uses a grid search to assess the best combination of learning rate, tree depth, and bag fraction based on cross-validated deviance. If a particular combination of parameters leads to an unconverged model, the script attempts again using slightly different parameters. Its output is any or all of: a table with deviance of evaluated models; all evaluated models; and/or the single model with the lowest deviance.

Usage

```
trainBRT(
  data,
  resp = names(data)[1],
  preds = names(data)[2:ncol(data)],
  learningRate = c(1e-04, 0.001, 0.01),
  treeComplexity = c(5, 3, 1),
  bagFraction = 0.6,
  minTrees = 1000,
  maxTrees = 8000,
  tries = 5,
  tryBy = c("learningRate", "treeComplexity", "maxTrees", "stepSize"),
  w = TRUE,
  anyway = FALSE,
  family = "bernoulli",
  out = "model",
  cores = 1,
  verbose = FALSE,
  ...
)
```

Arguments

data	Data frame.
resp	Response variable. This is either the name of the column in data or an integer indicating the column in data that has the response variable. The default is to use the first column in data as the response.
preds	Character vector or integer vector. Names of columns or column indices of predictors. The default is to use the second and subsequent columns in data.
learningRate	Numeric. Learning rate at which model learns from successive trees (Elith et al. 2008 recommend 0.0001 to 0.1).
treeComplexity	Positive integer. Tree complexity: depth of branches in a single tree (1 to 16).

bagFraction	Numeric in the range [0, 1]. Bag fraction: proportion of data used for training in cross-validation (Elith et al. 2008 recommend 0.5 to 0.7).
minTrees	Positive integer. Minimum number of trees to be scored as a "usable" model (Elith et al. 2008 recommend at least 1000). Default is 1000.
maxTrees	Positive integer. Maximum number of trees in model set.
tries	Integer > 0. Number of times to try to train a model with a particular set of tuning parameters. The function will stop training the first time a model converges (usually on the first attempt). Non-convergence seems to be related to the number of trees tried in each step. So if non-convergence occurs then the function automatically increases the number of trees in the step size until tries is reached.
tryBy	Character vector. A list that contains one or more of 'learningRate', 'treeComplexity', numTrees, and/or 'stepSize'. If a given combination of learningRate, treeComplexity, numTrees, stepSize, and bagFraction do not allow model convergence then then the function tries again but with alterations to any of the arguments named in tryBy: * learningRate: Decrease the learning rate by a factor of 10. * treeComplexity: Randomly increase/decrease tree complexity by 1 (minimum of 1). * maxTrees: Increase number of trees by 20 * stepSize: Increase step size (argument n.trees in gbm.step()) by 50 If tryBy is NULL then the function attempts to train the model with the same parameters up to tries times.
w	Weights. Any of: • TRUE: Causes the total weight of presences to equal the total weight of absences (if family='binomial') • FALSE: Each datum is assigned a weight of 1. • A numeric vector of weights, one per row in data. • The name of the column in data that contains site weights.
anyway	Logical. If FALSE (default), it is possible for no models to be returned if none converge and/or none had a number of trees is >= minTrees). If TRUE then all models are returned but with a warning.
family	Character. Name of error family.
out	Character vector. One or more values: • 'model': Model with the lowest deviance. • 'models': All models evaluated, sorted from lowest to highest deviance. • 'tuning': Data frame with tuning parameters, one row per model, sorted by deviance.
cores	Integer >= 1. Number of cores to use when calculating multiple models. Default is 1. If you have issues when cores > 1, please see the troubleshooting_parallel_operations guide.
verbose	Logical. If TRUE display progress.
...	Additional arguments (not used).

Value

The object that is returned depends on the value of the out argument. It can be a model object, a data frame, a list of models, or a list of two or more of these.

References

Elith, J., J.R. Leathwick, & T. Hastie. 2008. A working guide to boosted regression trees. *Journal of Animal Ecology* 77:802-813. doi:10.1111/j.13652656.2008.01390.x

See Also

gbm

Examples

```
# NB: The examples below show a very basic modeling workflow. They have been
# designed to work fast, not produce accurate, defensible models. They can
# take a few minutes to run.

library(mgcv)
library(sf)
library(terra)
set.seed(123)

### setup data
###############

# environmental rasters
rastFile <- system.file('extdata/madClim.tif', package='enmSdmX')
madClim <- rast(rastFile)

# coordinate reference system
wgs84 <- getCRS('WGS84')

# lemur occurrence data
data(lemurs)
occs <- lemurs[lemurs$species == 'Eulemur fulvus', ]
occs <- vect(occs, geom=c('longitude', 'latitude'), crs=wgs84)

occs <- elimCellDuplicates(occs, madClim)

occEnv <- extract(madClim, occs, ID = FALSE)
occEnv <- occEnv[complete.cases(occEnv), ]

# create 10000 background sites (or as many as raster can support)
bgEnv <- terra::spatSample(madClim, 20000)
bgEnv <- bgEnv[complete.cases(bgEnv), ]
bgEnv <- bgEnv[1:min(10000, nrow(bgEnv)), ]

# collate occurrences and background sites
presBg <- data.frame(
  presBg = c(
    rep(1, nrow(occEnv)),
    rep(0, nrow(bgEnv))
  )
)
```

```
env <- rbind(occEnv, bgEnv)
env <- cbind(presBg, env)

predictors <- c('bio1', 'bio12')

### calibrate models
####################

# Note that all of the trainXYZ functions can made to go faster using the
# "cores" argument (set to just 1, by default). The examples below will not
# go too much faster using more cores because they are simplified, but
# you can try!
cores <- 1

# MaxEnt
mx <- trainMaxEnt(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)

# MaxNet
mn <- trainMaxNet(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)

# generalized linear model (GLM)
gl <- trainGLM(
 data = env,
 resp = 'presBg',
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 verbose = TRUE,
 cores = cores
)

# generalized additive model (GAM)
ga <- trainGAM(
 data = env,
 resp = 'presBg',
 preds = predictors,
 verbose = TRUE,
 cores = cores
)
```

```
# natural splines
ns <- trainNS(
 data = env,
 resp = 'presBg',
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 df = 1:2, # too few values for reliable model(?)
 verbose = TRUE,
 cores = cores
)

# boosted regression trees
envSub <- env[1:1049, ] # subsetting data to run faster
brt <- trainBRT(
 data = envSub,
 resp = 'presBg',
 preds = predictors,
 learningRate = 0.001, # too few values for reliable model(?)
 treeComplexity = c(2, 3), # too few values for reliable model, but fast
 minTrees = 1200, # minimum trees for reliable model(?), but fast
 maxTrees = 1200, # too small for reliable model(?), but fast
 tryBy = 'treeComplexity',
 anyway = TRUE, # return models that did not converge
 verbose = TRUE,
 cores = cores
)

# random forests
rf <- trainRF(
 data = env,
 resp = 'presBg',
 preds = predictors,
 numTrees = c(100, 500), # using at least 500 recommended, but fast!
 verbose = TRUE,
 cores = cores
)

### make maps of models
#######################

# NB We do not have to scale rasters before predicting GLMs and NSs because we
# used the `scale = TRUE` argument in trainGLM() and trainNS().

mxMap <- predictEnmSdm(mx, madClim)
mnMap <- predictEnmSdm(mn, madClim)
glMap <- predictEnmSdm(gl, madClim)
gaMap <- predictEnmSdm(ga, madClim)
nsMap <- predictEnmSdm(ns, madClim)
brtMap <- predictEnmSdm(brt, madClim)
rfMap <- predictEnmSdm(rf, madClim)

maps <- c(
```

```
 mxMap,
 mnMap,
 glMap,
 gaMap,
 nsMap,
 brtMap,
 rfMap
)

names(maps) <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NSs', 'BRTs', 'RFs')
fun <- function() plot(occs, col='black', pch=3, add=TRUE)
plot(maps, fun = fun, nc = 4)

### compare model responses to BIO12 (mean annual precipitation)
#####################################################################

# make a data frame holding all other variables at mean across occurrences,
# varying only BIO12
occEnvMeans <- colMeans(occEnv, na.rm=TRUE)
occEnvMeans <- rbind(occEnvMeans)
occEnvMeans <- as.data.frame(occEnvMeans)
climFrame <- occEnvMeans[rep(1, 100), ]
rownames(climFrame) <- NULL

minBio12 <- min(env$bio12)
maxBio12 <- max(env$bio12)
climFrame$bio12 <- seq(minBio12, maxBio12, length.out=100)

predMx <- predictEnmSdm(mx, climFrame)
predMn <- predictEnmSdm(mn, climFrame)
predGl <- predictEnmSdm(gl, climFrame)
predGa <- predictEnmSdm(ga, climFrame)
predNat <- predictEnmSdm(ns, climFrame)
predBrt <- predictEnmSdm(brt, climFrame)
predRf <- predictEnmSdm(rf, climFrame)


plot(climFrame$bio12, predMx,
xlab='BIO12', ylab='Prediction', type='l', ylim=c(0, 1))

lines(climFrame$bio12, predMn, lty='solid', col='red')
lines(climFrame$bio12, predGl, lty='dotted', col='blue')
lines(climFrame$bio12, predGa, lty='dashed', col='green')
lines(climFrame$bio12, predNat, lty=4, col='purple')
lines(climFrame$bio12, predBrt, lty=5, col='orange')
lines(climFrame$bio12, predRf, lty=6, col='cyan')

legend(
   'topleft',
   inset = 0.01,
   legend = c(
 'MaxEnt',
 'MaxNet',
```

```
 'GLM',
 'GAM',
 'NS',
 'BRT',
 'RF'
   ),
   lty = c(1, 1:6),
   col = c(
 'black',
 'red',
 'blue',
 'green',
 'purple',
 'orange',
 'cyan'
   ),
   bg = 'white'
)

### compare response curves ###
###############################

modelNames <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NS', 'BRT', 'RF')

responses <- responseCurves(
 models = list(mx, mn, gl, ga, ns, brt, rf),
 env = bgEnv,
 ref = occEnv,
 vars = predictors,
 modelNames = modelNames
)

print(responses)
```

`trainByCrossValid` *Calibrate a distribution/niche model using cross-validation*

Description

This function is an extension of any of the `trainXYZ` functions for calibrating species distribution and ecological niche models. This function uses the `trainXYZ` function to calibrate and evaluate a suite of models using cross-validation. The models are evaluated against withheld data to determine the optimal settings for a "final" model using all available data. The function returns a set of models and/or a table with statistics on each model. The statistics represent various measures of model accuracy, and are calculated against training and test sites (separately).

Usage

```
trainByCrossValid(
  data,
  resp = names(data)[1],
  preds = names(data)[2:ncol(data)],
  folds = predicts::folds(data),
  trainFx = enmSdmX::trainGLM,
  ...,
  weightEvalTrain = TRUE,
  weightEvalTest = TRUE,
  na.rm = FALSE,
  outputModels = TRUE,
  verbose = 0
)
```

Arguments

`data`	Data frame or matrix. Response variable and environmental predictors (and no other fields) for presences and non-presence sites.
`resp`	Character or integer. Name or column index of response variable. Default is to use the first column in `data`.
`preds`	Character vector or integer vector. Names of columns or column indices of predictors. Default is to use the second and subsequent columns in `data` as predictors.
`folds`	Either a numeric vector, or matrix or data frame which specify which rows in `data` belong to which folds: • If a vector, there must be one value per row in `data`. If there are K unique values in the vector, then K unique models will be trained. Each model will use all of the data except for rows that match a particular value in the `folds` vector. For example, if `folds = c(1, 1, 1, 2, 2, 2, 3, 3, 3)`, then three models will be trained, one with all rows that match the 2s and 3s, one with all rows matching 1s and 2s, and one will all rows matching 1s and 3s. The models will be evaluated against the training data and against the withheld data. Use `NA` to exclude rows from all testing/training. The default is to construct 5 folds of roughly equal size. • If a matrix or data frame, there must be one row per row in `data`. Each column corresponds to a different model to be trained. For a given column there should be only two unique values, plus possibly `NA`s. Of the two values, the lesser value will be used to identify the calibration data and the greater value the evaluation data. Rows with `NA`s will be ignored and not used in training or testing. For example, a particular column could contain 1s, 2, and `NA`s. Data rows corresponding to 1s will be used as training data, data rows corresponding to 2s as test data, and rows with `NA` are dropped. The `NA` flag is useful for creating spatially-structured cross-validation folds where training and test sites are separated (spatially) by censored (ignored) data.

trainFx — Function, name of the trainXYZ function to use. Currently the functions/algorithms supported are trainBRT, trainGAM, trainGLM, trainMaxEnt, trainRF, and trainNS.

... — Arguments to pass to the "trainXYZ" function.

weightEvalTrain — Logical, if TRUE (default) and an argument named w is specified in ..., then evaluation statistics that support weighting will use the weights specified by w *for the "train" version of evaluation statistics*. If FALSE, there will be no weighting of sites. Note that this applies *only* to the calculation of evaluation statistics, not to model calibration. If w is supplied, they will be used for model calibration.

weightEvalTest — Logical, if TRUE (default) and an argument named w is specified in ..., then evaluation statistics that support weighting will use the weights specified by w *for the "test" version of evaluation statistics*. If FALSE, there will be no weighting of sites. Note that this applies *only* to the calculation of evaluation statistics. If w is supplied, they will be used for model calibration.

na.rm — Logical, if TRUE then remove NA predictions before calculating evaluation statistics. If FALSE (default), propagate NAs (meaning if predictions contain NAs, then the evaluation statistic will most likely also be NA.)

outputModels — If TRUE, then return all models (in addition to tables reporting tuning paramaeters and evaluation metrics). *WARNING*: Depending on the type of model and amount of data, retuning all models may produce objects that are very large in memory.

verbose — Numeric. If 0 show no progress updates. If > 0 then show minimal progress updates for this function only. If > 1 show detailed progress for this function. If > 2 show detailed progress plus detailed progress for the trainXYZ function.

Details

In some cases models do not converge (e.g., boosted regression trees and generalized additive models sometimes suffer from this issue). In this case the model will be skipped, but a data frame with the k-fold and model number in the fold will be returned in the $meta element in the output. If no models converged, then this data frame will be empty.

Value

A list object with several named elements:

- meta: Meta-data on the model call.
- folds: The folds object.
- models (if outputModels is TRUE): A list of model objects, one per data fold.
- tuning: One data frame per k-fold, each containing evaluation statistics for all candidate models in the fold. In addition to algorithm-specific fields, these consist of:
  - 'logLoss': Log loss. Higher (less negative) values imply better fit.
  - 'cbi': Continuous Boyce Index (CBI). Calculated with evalContBoyce.
  - 'auc': Area under the receiver-operator characteristic curve (AUC). Calculated with evalAUC.

- 'tss': Maximum value of the True Skill Statistic. Calculated with evalTSS.
- 'msss': Sensitivity and specificity calculated at the threshold that maximizes sensitivity (true presence prediction rate) plus specificity (true absence prediction rate).
- 'mdss': Sensitivity (se) and specificity (sp) calculated at the threshold that minimizes the difference between sensitivity and specificity.
- 'minTrainPres': Sensitivity (se) and specificity (sp) at the greatest threshold at which all training presences are classified as "present".
- 'trainSe95' and/or 'trainSe90': Sensitivity (se) and specificity (sp) at the threshold that ensures either 95 or 90 percent of all training presences are classified as "present" (training sensitivity = 0.95 or 0.9).

References

Fielding, A.H. and J.F. Bell. 1997. A review of methods for the assessment of prediction errors in conservation presence/absence models. *Environmental Conservation* 24:38-49. doi:10.1017/S0376892997000088 La Rest, K., Pinaud, D., Monestiez, P., Chadoeuf, J., and Bretagnolle, V. 2014. Spatial leave-one-out cross-validation for variable selection in the presence of spatial autocorrelation. *Global Ecology and Biogeography* 23:811-820. doi:10.1111/geb.12161 Radosavljevic, A. and Anderson, R.P. 2014. Making better Maxent models of species distributions: complexity, overfitting and evaluation. *Journal of Biogeography* 41:629-643. doi:10.1111/jbi.12227

See Also

summaryByCrossValid, trainBRT, trainGAM, trainGLM, trainMaxEnt, trainMaxNet, trainNS, trainRF

Examples

```
# The example below show a very basic modeling workflow. It has been
# designed to work fast, not produce accurate, defensible models.
# The general idea is to calibrate a series of models and evaluate them
# against a withheld set of data. One can then use the series of models
# of the top models to better select a "final" model.

## Not run:
# Running the entire set of commands can take a few minutes. This can
# be sped up by increasing the number of cores used. The examples below use
# one core, but you can change that argument according to your machine's
# capabilities.

library(sf)
library(terra)
set.seed(123)

### setup data
###############

# environmental rasters
rastFile <- system.file('extdata/madClim.tif', package='enmSdmX')
madClim <- rast(rastFile)
```

```
# coordinate reference system
wgs84 <- getCRS('WGS84')

# lemur occurrence data
data(lemurs)
occs <- lemurs[lemurs$species == 'Eulemur fulvus', ]
occs <- vect(occs, geom=c('longitude', 'latitude'), crs=wgs84)

occs <- elimCellDuplicates(occs, madClim)

occEnv <- extract(madClim, occs, ID = FALSE)
occEnv <- occEnv[complete.cases(occEnv), ]

# create background sites (using just 1000 to speed things up!)
bgEnv <- terra::spatSample(madClim, 3000)
bgEnv <- bgEnv[complete.cases(bgEnv), ]
bgEnv <- bgEnv[sample(nrow(bgEnv), 1000), ]

# collate occurrences and background sites
presBg <- data.frame(
   presBg = c(
      rep(1, nrow(occEnv)),
      rep(0, nrow(bgEnv))
   )
)

env <- rbind(occEnv, bgEnv)
env <- cbind(presBg, env)

predictors <- c('bio1', 'bio12')

# using "vector" form of "folds" argument
folds <- predicts::kfold(env, 3) # just 3 folds (for speed)

### calibrate models
####################

cores <- 1 # increase this to go faster, if your computer handles it

## MaxEnt
mxx <- trainByCrossValid(
 data = env,
 resp = 'presBg',
 preds = c('bio1', 'bio12'),
 folds = folds,
 trainFx = trainMaxEnt,
 regMult = 1:2, # too few values for valid model, but fast!
 verbose = 1,
 cores = cores
)

# summarize MaxEnt feature sets and regularization across folds
```

```
summaryByCrossValid(mxx)

## MaxNet
mnx <- trainByCrossValid(
 data = env,
 resp = 'presBg',
 preds = c('bio1', 'bio12'),
 folds = folds,
 trainFx = trainMaxNet,
 regMult = 1:2, # too few values for valid model, but fast!
 verbose = 1,
 cores = cores
)

# summarize MaxEnt feature sets and regularization across folds
summaryByCrossValid(mnx)

## generalized linear models
glx <- trainByCrossValid(
 data = env,
 resp = 'presBg',
 preds = c('bio1', 'bio12'),
 folds = folds,
 trainFx = trainGLM,
 verbose = 1,
 cores = cores
)

# summarize GLM terms in best models
summaryByCrossValid(glx)

## generalized additive models
gax <- trainByCrossValid(
 data = env,
 resp = 'presBg',
 preds = c('bio1', 'bio12'),
 folds = folds,
 trainFx = trainGAM,
 verbose = 1,
 cores = cores
)

# summarize GAM terms in best models
summaryByCrossValid(gax)

## natural splines
nsx <- trainByCrossValid(
 data = env,
 resp = 'presBg',
 preds = c('bio1', 'bio12'),
 folds = folds,
 trainFx = trainNS,
 df = 1:2,
```

```
 verbose = 1,
 cores = cores
)

# summarize NS terms in best models
summaryByCrossValid(nsx)

## boosted regression trees
brtx <- trainByCrossValid(
 data = env,
 resp = 'presBg',
 preds = c('bio1', 'bio12'),
 folds = folds,
 trainFx = trainBRT,
 learningRate = c(0.001, 0.0001), # too few values for reliable model(?)
 treeComplexity = c(2, 4), # too few values for reliable model, but fast
 minTrees = 1000,
 maxTrees = 1500, # too small for reliable model(?), but fast
 tryBy = 'treeComplexity',
 anyway = TRUE, # return models that did not converge
 verbose = 1,
 cores = cores
)

# summarize BRT parameters across best models
summaryByCrossValid(brtx)

## random forests
rfx <- trainByCrossValid(
 data = env,
 resp = 'presBg',
 preds = c('bio1', 'bio12'),
 folds = folds,
 trainFx = trainRF,
 verbose = 1,
 cores = cores
)

# summarize RF parameters in best models
summaryByCrossValid(rfx)


## End(Not run)
```

`trainESM` *Calibrate an ensemble of small models*

Description

This function calibrates a set of "ensembles of small models" (ESM), which are designed for modeling species with few occurrence records. In the original formulation, each model has two covariates

interacting additively. Models are calibrated using all possible combinations of covariates. By default, this function does the same, but can also include univariate models, models with two covariates plus their interaction term, and models with quadratic and corresponding linear terms. This function will *only* train generalized linear models. Extending the types of algorithms is planned!

Usage

```
trainESM(
  data,
  resp = names(data)[1],
  preds = names(data)[2:ncol(data)],
  univariate = FALSE,
  quadratic = FALSE,
  interaction = FALSE,
  interceptOnly = FALSE,
  method = "glm.fit",
  scale = NA,
  w = TRUE,
  family = stats::binomial(),
  ...,
  verbose = FALSE
)
```

Arguments

`data`	Data frame or matrix. Response variable and environmental predictors (and no other fields) for presences and non-presence sites.
`resp`	Character or integer. Name or column index of response variable. Default is to use the first column in `data`.
`preds`	Character vector or integer vector. Names of columns or column indices of predictors. Default is to use the second and subsequent columns in `data` as predictors.
`univariate`, `quadratic`, `interaction`	`TRUE` or `FALSE`: Whether or not to include univariate models, quadratic models, and/or models with 2-way interactions (default is `FALSE`).
`interceptOnly`	If `TRUE`, include an intercept-only model (default is `FALSE`).
`method`	Character: Name of function used to solve the GLM. For "normal" GLMs, this can be `'glm.fit'` (default), `'brglmFit'` (from the **brglm2** package), or another function.
`scale`	Either `NA` (default), or `TRUE` or `FALSE`. If `TRUE`, the predictors will be centered and scaled by dividing by subtracting their means then dividing by their standard deviations. The means and standard deviations will be returned in the model object under an element named "`scales`". For example, if you do something like `model <- trainGLM(data, scale=TRUE)`, then you can get the means and standard deviations using `model$scales$mean` and `model$scales$sd`. If `FALSE`, no scaling is done. If `NA` (default), then the function will check to see if non-factor predictors have means ~0 and standard deviations ~1. If not, then a warning will be printed, but the function will continue to do its operations.

w	Weights. Any of: • TRUE: Causes the total weight of presences to equal the total weight of absences (if family='binomial') • FALSE: Each datum is assigned a weight of 1. • A numeric vector of weights, one per row in data. • The name of the column in data that contains site weights.
family	Character or function. Name of family for data error structure (see family). Default is to use the 'binomial' family.
...	Arguments to pass to glm
verbose	Logical. If TRUE then display progress.

Value

A list object with several named elements:

- models: A list with each ESM model.
- tuning: A data.frame with one row per model, in the order as they appear in $models.

References

Breiner, F.T., Guisan, A., Bergamini, A., and Nobis, M.P. 2015. Overcoming limitations of modeling rare species by using ensembles of small models. *Methods in Ecology and Evolution* 6:1210-1218.. doi:10.1111/2041210X.12403 Lomba, A., L. Pellissier, C. Randin, J. Vicente, J. Horondo, and A. Guisan. 2010. Overcoming the rare species modeling complex: A novel hierarchical framework applied to an Iberian endemic plant. *Biological Conservation* 143:2647-2657. doi:10.1016/j.biocon.2010.07.007

See Also

trainBRT, trainGAM, trainGLM, trainMaxEnt, trainMaxNet, trainNS, trainRF, trainByCrossValid

Examples

```
# NB: The examples below show a very basic modeling workflow. They have been
# designed to work fast, not produce accurate, defensible models. They can
# take a few minutes to run.

library(terra)
set.seed(123)

### setup data
###############

# environmental rasters
rastFile <- system.file('extdata/madClim.tif', package='enmSdmX')
madClim <- rast(rastFile)

# coordinate reference system
wgs84 <- getCRS('WGS84')
```

```
# lemur occurrence data
data(lemurs)
occs <- lemurs[lemurs$species == 'Eulemur fulvus', ]
occs <- vect(occs, geom=c('longitude', 'latitude'), crs=wgs84)

occs <- elimCellDuplicates(occs, madClim)

occEnv <- extract(madClim, occs, ID = FALSE)
occEnv <- occEnv[complete.cases(occEnv), ]

# create 10000 background sites (or as many as raster can support)
bgEnv <- terra::spatSample(madClim, 20000)
bgEnv <- bgEnv[complete.cases(bgEnv), ]
bgEnv <- bgEnv[1:min(10000, nrow(bgEnv)), ]

# collate occurrences and background sites
presBg <- data.frame(
  presBg = c(
    rep(1, nrow(occEnv)),
    rep(0, nrow(bgEnv))
  )
)

env <- rbind(occEnv, bgEnv)
env <- cbind(presBg, env)

predictors <- c('bio1', 'bio12')

### calibrate models
####################

# "traditional" ESMs with just 2 linear predictors
# just one model in this case because we have just 2 predictors
esm1 <- trainESM(
   data = env,
   resp = 'presBg',
   preds = predictors,
   family = stats::binomial(),
   scale = TRUE,
   w = TRUE
)

str(esm1, 1)
esm1$tuning

# extended ESM with other kinds of terms
esm2 <- trainESM(
   data = env,
   resp = 'presBg',
   preds = predictors,
   univariate = TRUE,
   quadratic = TRUE,
```

```
   interaction = TRUE,
   interceptOnly = TRUE,
   family = stats::binomial(),
   scale = TRUE,
   w = TRUE,
   verbose = TRUE
)

str(esm2, 1)
esm2$tuning

### make a set of predictions to rasters
########################################

# center environmental rasters and divide by their SD
madClimScaled <- scale(madClim, center = esm2$scale$mean, scale = esm2$scale$sd)

# make one raster per model
predictions <- list()
for (i in 1:length(esm2$models)) {
    predictions[[i]] <- predict(madClimScaled, esm2$models[[i]], type = 'response')
}

# combine into a "stack"
predictions <- do.call(c, predictions)
names(predictions) <- esm2$tuning$model
plot(predictions)

# calculate (unweighted) mean
prediction <- mean(predictions)
plot(prediction)
plot(occs, pch = 1, add = TRUE)
```

`trainGAM` *Calibrate a generalized additive model (GAM)*

Description

This function constructs a generalized additive model. By default, the model is constructed in a two-stage process. First, the "construct" phase generates a series of simple models with univariate and bivariate interaction terms. These simple models are then ranked based on their AICc. Second, the "select" phase creates a "full" model from the simple models such that there is at least `presPerTermInitial` presences (if the response is binary) or data rows (if not) for each smooth term to be estimated (not counting the intercept). Finally, it selects the best model using AICc from all possible subsets of this "full" model. Its output is any or all of: a table with AICc for all evaluated models; all models evaluated in the "selection" phase; and/or the single model with the lowest AICc.

Usage

```
trainGAM(
  data,
  resp = names(data)[1],
  preds = names(data)[2:ncol(data)],
  gamma = 1,
  scale = 0,
  smoothingBasis = "cs",
  interaction = "te",
  interceptOnly = TRUE,
  construct = TRUE,
  select = TRUE,
  presPerTermInitial = 10,
  presPerTermFinal = 10,
  maxTerms = 8,
  w = TRUE,
  family = "binomial",
  out = "model",
  cores = 1,
  verbose = FALSE,
  ...
)
```

Arguments

data	Data frame.
resp	Response variable. This is either the name of the column in data or an integer indicating the column in data that has the response variable. The default is to use the first column in data as the response.
preds	Character vector or integer vector. Names of columns or column indices of predictors. The default is to use the second and subsequent columns in data.
gamma	Initial penalty to degrees of freedom to use (larger ==> smoother fits).
scale	A numeric value indicating the "scale" parameter (see argument scale in gam). The default is 0 (which allows a single smoother for Poisson and binomial error families and unknown scale for all others.)
smoothingBasis	Character. Indicates the type of smoothing basis. The default is 'cs' (cubic splines), but see smooth.terms for other options. This is the value of argument bs in a s function.
interaction	Character or NULL. Type of interaction term to use (te, ts, s, etc.). See ?te (for example) for help on any one of these. If NULL, then interactions are not used.
interceptOnly	If TRUE (default) and model selection is enabled, then include an intercept-only model.
construct	If TRUE (default), then construct the model by computing AICc for all univariate and bivariate models. Then add terms up to maximum set by presPerTermInitial and maxTerms.

`select` If `TRUE` (default), then calculate AICc for all possible subsets of models and return the model with the lowest AICc of these. This step if performed *after* model construction (if `construct` is `TRUE`).

`presPerTermInitial`
Positive integer. Minimum number of presences needed per model term for a term to be included in the model construction stage. Used only if `construct` is `TRUE`.

`presPerTermFinal`
Positive integer. Minimum number of presence sites per term in initial starting model; used only if `select` is `TRUE`.

`maxTerms` Maximum number of terms to be used in any model, not including the intercept (default is 8). Used only if `construct` is `TRUE`.

`w` Weights. Any of:

- `TRUE`: Causes the total weight of presences to equal the total weight of absences (if `family='binomial'`)
- `FALSE`: Each datum is assigned a weight of 1.
- A numeric vector of weights, one per row in `data`.
- The name of the column in `data` that contains site weights.

`family` Name of family for data error structure (see `?family`).

`out` Character vector. One or more values:

- `'model'`: Model with the lowest AICc.
- `'models'`: All models evaluated, sorted from lowest to highest AICc (lowest is best).
- `'tuning'`: Data frame with tuning parameters, one row per model, sorted by AICc.

`cores` Integer >= 1. Number of cores to use when calculating multiple models. Default is 1. If you have issues when `cores` > 1, please see the `troubleshooting_parallel_operations` guide.

`verbose` Logical. If `TRUE` then display intermediate results on the display device.

`...` Extra arguments (not used).

Value

The object that is returned depends on the value of the `out` argument. It can be a model object, a data frame, a list of models, or a list of all two or more of these.

See Also

`gam`

Examples

```
# NB: The examples below show a very basic modeling workflow. They have been
# designed to work fast, not produce accurate, defensible models. They can
# take a few minutes to run.
```

```
library(mgcv)
library(sf)
library(terra)
set.seed(123)

### setup data
###############

# environmental rasters
rastFile <- system.file('extdata/madClim.tif', package='enmSdmX')
madClim <- rast(rastFile)

# coordinate reference system
wgs84 <- getCRS('WGS84')

# lemur occurrence data
data(lemurs)
occs <- lemurs[lemurs$species == 'Eulemur fulvus', ]
occs <- vect(occs, geom=c('longitude', 'latitude'), crs=wgs84)

occs <- elimCellDuplicates(occs, madClim)

occEnv <- extract(madClim, occs, ID = FALSE)
occEnv <- occEnv[complete.cases(occEnv), ]

# create 10000 background sites (or as many as raster can support)
bgEnv <- terra::spatSample(madClim, 20000)
bgEnv <- bgEnv[complete.cases(bgEnv), ]
bgEnv <- bgEnv[1:min(10000, nrow(bgEnv)), ]

# collate occurrences and background sites
presBg <- data.frame(
  presBg = c(
    rep(1, nrow(occEnv)),
    rep(0, nrow(bgEnv))
  )
)

env <- rbind(occEnv, bgEnv)
env <- cbind(presBg, env)

predictors <- c('bio1', 'bio12')

### calibrate models
####################

# Note that all of the trainXYZ functions can made to go faster using the
# "cores" argument (set to just 1, by default). The examples below will not
# go too much faster using more cores because they are simplified, but
# you can try!
cores <- 1
```

```
# MaxEnt
mx <- trainMaxEnt(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)

# MaxNet
mn <- trainMaxNet(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)

# generalized linear model (GLM)
gl <- trainGLM(
 data = env,
 resp = 'presBg',
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 verbose = TRUE,
 cores = cores
)

# generalized additive model (GAM)
ga <- trainGAM(
 data = env,
 resp = 'presBg',
 preds = predictors,
 verbose = TRUE,
 cores = cores
)

# natural splines
ns <- trainNS(
 data = env,
 resp = 'presBg',
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 df = 1:2, # too few values for reliable model(?)
 verbose = TRUE,
 cores = cores
)

# boosted regression trees
envSub <- env[1:1049, ] # subsetting data to run faster
brt <- trainBRT(
```

```
 data = envSub,
 resp = 'presBg',
 preds = predictors,
 learningRate = 0.001, # too few values for reliable model(?)
 treeComplexity = c(2, 3), # too few values for reliable model, but fast
 minTrees = 1200, # minimum trees for reliable model(?), but fast
 maxTrees = 1200, # too small for reliable model(?), but fast
 tryBy = 'treeComplexity',
 anyway = TRUE, # return models that did not converge
 verbose = TRUE,
 cores = cores
)

# random forests
rf <- trainRF(
 data = env,
 resp = 'presBg',
 preds = predictors,
 numTrees = c(100, 500), # using at least 500 recommended, but fast!
 verbose = TRUE,
 cores = cores
)

### make maps of models
#######################

# NB We do not have to scale rasters before predicting GLMs and NSs because we
# used the `scale = TRUE` argument in trainGLM() and trainNS().

mxMap <- predictEnmSdm(mx, madClim)
mnMap <- predictEnmSdm(mn, madClim)
glMap <- predictEnmSdm(gl, madClim)
gaMap <- predictEnmSdm(ga, madClim)
nsMap <- predictEnmSdm(ns, madClim)
brtMap <- predictEnmSdm(brt, madClim)
rfMap <- predictEnmSdm(rf, madClim)

maps <- c(
 mxMap,
 mnMap,
 glMap,
 gaMap,
 nsMap,
 brtMap,
 rfMap
)

names(maps) <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NSs', 'BRTs', 'RFs')
fun <- function() plot(occs, col='black', pch=3, add=TRUE)
plot(maps, fun = fun, nc = 4)

### compare model responses to BIO12 (mean annual precipitation)
################################################################
```

```
# make a data frame holding all other variables at mean across occurrences,
# varying only BIO12
occEnvMeans <- colMeans(occEnv, na.rm=TRUE)
occEnvMeans <- rbind(occEnvMeans)
occEnvMeans <- as.data.frame(occEnvMeans)
climFrame <- occEnvMeans[rep(1, 100), ]
rownames(climFrame) <- NULL

minBio12 <- min(env$bio12)
maxBio12 <- max(env$bio12)
climFrame$bio12 <- seq(minBio12, maxBio12, length.out=100)

predMx <- predictEnmSdm(mx, climFrame)
predMn <- predictEnmSdm(mn, climFrame)
predGl <- predictEnmSdm(gl, climFrame)
predGa <- predictEnmSdm(ga, climFrame)
predNat <- predictEnmSdm(ns, climFrame)
predBrt <- predictEnmSdm(brt, climFrame)
predRf <- predictEnmSdm(rf, climFrame)


plot(climFrame$bio12, predMx,
xlab='BIO12', ylab='Prediction', type='l', ylim=c(0, 1))

lines(climFrame$bio12, predMn, lty='solid', col='red')
lines(climFrame$bio12, predGl, lty='dotted', col='blue')
lines(climFrame$bio12, predGa, lty='dashed', col='green')
lines(climFrame$bio12, predNat, lty=4, col='purple')
lines(climFrame$bio12, predBrt, lty=5, col='orange')
lines(climFrame$bio12, predRf, lty=6, col='cyan')

legend(
   'topleft',
   inset = 0.01,
   legend = c(
 'MaxEnt',
 'MaxNet',
 'GLM',
 'GAM',
 'NS',
 'BRT',
 'RF'
   ),
   lty = c(1, 1:6),
   col = c(
 'black',
 'red',
 'blue',
 'green',
 'purple',
 'orange',
 'cyan'
```

```
   ),
   bg = 'white'
)

### compare response curves ###
###############################

modelNames <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NS', 'BRT', 'RF')

responses <- responseCurves(
 models = list(mx, mn, gl, ga, ns, brt, rf),
 env = bgEnv,
 ref = occEnv,
 vars = predictors,
 modelNames = modelNames
)

print(responses)
```

`trainGLM` *Calibrate a generalized linear model (GLM)*

Description

This function constructs a generalized linear model. By default, the model is constructed in a two-stage process. First, the "construct" phase generates a series of simple models with univariate, quadratic, or 2-way-interaction terms. These simple models are then ranked based on their AICc. Second, the "select" phase creates a "full" model from the simple models such that there is at least `presPerTermInitial` presences (if the response is binary) or data rows (if not) for each coefficient to be estimated (not counting the intercept). Finally, it selects the best model using AICc from all possible subsets of this "full" model, while respecting marginality (i.e., all lower-order terms of higher-order terms appear in the model).

The function outputs any or all of: a table with AICc for all evaluated models; all models evaluated in the "selection" phase; and/or the single model with the lowest AICc.

Usage

```
trainGLM(
  data,
  resp = names(data)[1],
  preds = names(data)[2:ncol(data)],
  scale = NA,
  construct = TRUE,
  select = TRUE,
  quadratic = TRUE,
```

```
  interaction = TRUE,
  interceptOnly = TRUE,
  method = "glm.fit",
  presPerTermInitial = 10,
  presPerTermFinal = 10,
  maxTerms = 8,
  w = TRUE,
  family = stats::binomial(),
  removeInvalid = TRUE,
  failIfNoValid = TRUE,
  out = "model",
  cores = 1,
  verbose = FALSE,
  ...
)
```

Arguments

data	Data frame.
resp	Response variable. This is either the name of the column in data or an integer indicating the column in data that has the response variable. The default is to use the first column in data as the response.
preds	Character vector or integer vector. Names of columns or column indices of predictors. The default is to use the second and subsequent columns in data.
scale	Either NA (default), or TRUE or FALSE. If TRUE, the predictors will be centered and scaled by dividing by subtracting their means then dividing by their standard deviations. The means and standard deviations will be returned in the model object under an element named "scales". For example, if you do something like model <- trainGLM(data, scale=TRUE), then you can get the means and standard deviations using model$scales$mean and model$scales$sd. If FALSE, no scaling is done. If NA (default), then the function will check to see if non-factor predictors have means ~0 and standard deviations ~1. If not, then a warning will be printed, but the function will continue to do its operations.
construct	Logical. If TRUE (default) then construct model from individual terms entered in order from lowest to highest AICc up to limits set by presPerTermInitial or maxTerms is met. If FALSE then the "full" model consists of all terms allowed by quadratic and interaction.
select	Logical. If TRUE (default) then calculate AICc for all possible subsets of models and return the model with the lowest AICc of these. This step if performed *after* model construction (if any).
quadratic	Logical. Used only if construct is TRUE. If TRUE (default) then include quadratic terms in model construction stage for non-factor predictors.
interaction	Logical. Used only if construct is TRUE. If TRUE (default) then include 2-way interaction terms (including interactions between factor predictors).
interceptOnly	If TRUE (default) and model selection is enabled, then include an intercept-only model.

`method` Character: Name of function used to solve the GLM. For "normal" GLMs, this can be `'glm.fit'` (default), `'brglmFit'` (from the **brglm2** package), or another function.

`presPerTermInitial` Positive integer. Minimum number of presences needed per model term for a term to be included in the model construction stage. Used only is `construct` is TRUE.

`presPerTermFinal` Positive integer. Minimum number of presence sites per term in initial starting model. Used only if `select` is `TRUE`.

`maxTerms` Maximum number of terms to be used in any model, not including the intercept (default is 8). Used only if `construct` is `TRUE`.

`w` Weights. Any of:

- `TRUE`: Causes the total weight of presences to equal the total weight of absences (if `family='binomial'`)
- `FALSE`: Each datum is assigned a weight of 1.
- A numeric vector of weights, one per row in `data`.
- The name of the column in `data` that contains site weights.

`family` Name of family for data error structure (see `family`). Default is to use the 'binomial' family.

`removeInvalid` Logical. If `TRUE` (default), remove models that either did not converge or have parameter estimates near the boundaries (usually negative or positive infinity). If you run this function with 'construct = TRUE' (i.e., construct a "full" model from the best "small" models), then any small model that either did not converge or had parameters that are near the boundary (usually negative or positive infinity) are removed from consideration as terms in "full" model.

`failIfNoValid` Logical. If `TRUE` (default), and the "full" model either does not converge or has parameters near the boundary, then the function will fail. If `FALSE`, then return `NULL` in this case.

`out` Character vector. One or more values:

- `'model'`: Model with the lowest AICc.
- `'models'`: All models evaluated, sorted from lowest to highest AICc (lowest is best).
- `'tuning'`: Data frame with tuning parameters, one row per model, sorted by AICc.

`cores` Integer >= 1. Number of cores to use when calculating multiple models. Default is 1. If you have issues when `cores` > 1, please see the `troubleshooting_parallel_operations` guide.

`verbose` Logical. If `TRUE` then display progress.

`...` Arguments to pass to `glm`.

Details

This function is designed to find the most parsimonious model given the amount of calibration data that is available to it. 'trainGLM()' can work with any data, but has been designed to work specifically as a species distribution model where the response is either binary (default) or abundance.

Specifically, it 1) identifies the most parsimonious model (lowest AICc) with 2) optimal flexibility (optimal degrees of freedom in splines) and 3) allows for (but does not require) interaction terms between predictors (if desired). If the defaults are used, the following procedure is applied:

- Constructing a set of simple model terms, each with 1 to 4 degrees of freedom. Terms can be univariate or bilabiate (two-way interactions). Predictors can be continuous or factors. If any simple models has convergence issues or boundary issues (coefficients that approach negative or positive infinity), it is removed.
- Constructing a series of models, each with one of the terms, then using the models to rank terms by AICc.
- From the top set of terms, creating a "full" model. The full model will ensure the maximum number of terms is <= 'maxTerms', and that for each term, there are at least 'presPerTermFinal' data points.
- All possible submodels, plus the full model, are evaluated and ranked by AICc. If a model has convergence or boundary issues, it is removed from the set. The most parsimonious model (lowest AICc) is returned.

Value

The object that is returned depends on the value of the `out` argument. It can be a model object, a data frame, a list of models, or a list of all two or more of these. If `scale` is `TRUE`, any model object will also have an element named `$scale`, which contains the means and standard deviations for predictors that are not factors. The data frame reports the AICc for all of the models evaluated, sorted by best to worst. The `converged` column indicates whether the model converged ("`TRUE`" is good), and the `boundary` column whether the model parameters are near the boundary (usually, negative or positive infinity; "`FALSE`" is good).

See Also

`glm`

Examples

```
# NB: The examples below show a very basic modeling workflow. They have been
# designed to work fast, not produce accurate, defensible models. They can
# take a few minutes to run.

library(mgcv)
library(sf)
library(terra)
set.seed(123)

### setup data
###############

# environmental rasters
rastFile <- system.file('extdata/madClim.tif', package='enmSdmX')
madClim <- rast(rastFile)
```

```
# coordinate reference system
wgs84 <- getCRS('WGS84')

# lemur occurrence data
data(lemurs)
occs <- lemurs[lemurs$species == 'Eulemur fulvus', ]
occs <- vect(occs, geom=c('longitude', 'latitude'), crs=wgs84)

occs <- elimCellDuplicates(occs, madClim)

occEnv <- extract(madClim, occs, ID = FALSE)
occEnv <- occEnv[complete.cases(occEnv), ]

# create 10000 background sites (or as many as raster can support)
bgEnv <- terra::spatSample(madClim, 20000)
bgEnv <- bgEnv[complete.cases(bgEnv), ]
bgEnv <- bgEnv[1:min(10000, nrow(bgEnv)), ]

# collate occurrences and background sites
presBg <- data.frame(
  presBg = c(
    rep(1, nrow(occEnv)),
    rep(0, nrow(bgEnv))
  )
)

env <- rbind(occEnv, bgEnv)
env <- cbind(presBg, env)

predictors <- c('bio1', 'bio12')

### calibrate models
####################

# Note that all of the trainXYZ functions can made to go faster using the
# "cores" argument (set to just 1, by default). The examples below will not
# go too much faster using more cores because they are simplified, but
# you can try!
cores <- 1

# MaxEnt
mx <- trainMaxEnt(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)

# MaxNet
mn <- trainMaxNet(
 data = env,
```

```
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)

# generalized linear model (GLM)
gl <- trainGLM(
 data = env,
 resp = 'presBg',
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 verbose = TRUE,
 cores = cores
)

# generalized additive model (GAM)
ga <- trainGAM(
 data = env,
 resp = 'presBg',
 preds = predictors,
 verbose = TRUE,
 cores = cores
)

# natural splines
ns <- trainNS(
 data = env,
 resp = 'presBg',
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 df = 1:2, # too few values for reliable model(?)
 verbose = TRUE,
 cores = cores
)

# boosted regression trees
envSub <- env[1:1049, ] # subsetting data to run faster
brt <- trainBRT(
 data = envSub,
 resp = 'presBg',
 preds = predictors,
 learningRate = 0.001, # too few values for reliable model(?)
 treeComplexity = c(2, 3), # too few values for reliable model, but fast
 minTrees = 1200, # minimum trees for reliable model(?), but fast
 maxTrees = 1200, # too small for reliable model(?), but fast
 tryBy = 'treeComplexity',
 anyway = TRUE, # return models that did not converge
 verbose = TRUE,
 cores = cores
)
```

```
# random forests
rf <- trainRF(
 data = env,
 resp = 'presBg',
 preds = predictors,
 numTrees = c(100, 500), # using at least 500 recommended, but fast!
 verbose = TRUE,
 cores = cores
)

### make maps of models
#######################

# NB We do not have to scale rasters before predicting GLMs and NSs because we
# used the `scale = TRUE` argument in trainGLM() and trainNS().

mxMap <- predictEnmSdm(mx, madClim)
mnMap <- predictEnmSdm(mn, madClim)
glMap <- predictEnmSdm(gl, madClim)
gaMap <- predictEnmSdm(ga, madClim)
nsMap <- predictEnmSdm(ns, madClim)
brtMap <- predictEnmSdm(brt, madClim)
rfMap <- predictEnmSdm(rf, madClim)

maps <- c(
 mxMap,
 mnMap,
 glMap,
 gaMap,
 nsMap,
 brtMap,
 rfMap
)

names(maps) <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NSs', 'BRTs', 'RFs')
fun <- function() plot(occs, col='black', pch=3, add=TRUE)
plot(maps, fun = fun, nc = 4)

### compare model responses to BIO12 (mean annual precipitation)
################################################################

# make a data frame holding all other variables at mean across occurrences,
# varying only BIO12
occEnvMeans <- colMeans(occEnv, na.rm=TRUE)
occEnvMeans <- rbind(occEnvMeans)
occEnvMeans <- as.data.frame(occEnvMeans)
climFrame <- occEnvMeans[rep(1, 100), ]
rownames(climFrame) <- NULL

minBio12 <- min(env$bio12)
maxBio12 <- max(env$bio12)
climFrame$bio12 <- seq(minBio12, maxBio12, length.out=100)
```

```
predMx <- predictEnmSdm(mx, climFrame)
predMn <- predictEnmSdm(mn, climFrame)
predGl <- predictEnmSdm(gl, climFrame)
predGa <- predictEnmSdm(ga, climFrame)
predNat <- predictEnmSdm(ns, climFrame)
predBrt <- predictEnmSdm(brt, climFrame)
predRf <- predictEnmSdm(rf, climFrame)


plot(climFrame$bio12, predMx,
xlab='BIO12', ylab='Prediction', type='l', ylim=c(0, 1))

lines(climFrame$bio12, predMn, lty='solid', col='red')
lines(climFrame$bio12, predGl, lty='dotted', col='blue')
lines(climFrame$bio12, predGa, lty='dashed', col='green')
lines(climFrame$bio12, predNat, lty=4, col='purple')
lines(climFrame$bio12, predBrt, lty=5, col='orange')
lines(climFrame$bio12, predRf, lty=6, col='cyan')

legend(
   'topleft',
   inset = 0.01,
   legend = c(
 'MaxEnt',
 'MaxNet',
 'GLM',
 'GAM',
 'NS',
 'BRT',
 'RF'
   ),
   lty = c(1, 1:6),
   col = c(
 'black',
 'red',
 'blue',
 'green',
 'purple',
 'orange',
 'cyan'
   ),
   bg = 'white'
)

### compare response curves ###
###############################

modelNames <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NS', 'BRT', 'RF')

responses <- responseCurves(
 models = list(mx, mn, gl, ga, ns, brt, rf),
 env = bgEnv,
 ref = occEnv,
```

```
 vars = predictors,
 modelNames = modelNames
)

print(responses)
```

`trainMaxEnt` *Calibrate a MaxEnt model using AICc*

Description

This function calculates the "best" MaxEnt model using AICc across all possible combinations of a set of master regularization parameters and feature classes. The best model has the lowest AICc, with ties broken by number of features (fewer is better), regularization multiplier (higher better), then finally the number of coefficients (fewer better).

Usage

```
trainMaxEnt(
  data,
  resp = names(data)[1],
  preds = names(data)[2:ncol(data)],
  regMult = c(seq(0.5, 5, by = 0.5), 7.5, 10),
  classes = "default",
  testClasses = TRUE,
  dropOverparam = TRUE,
  anyway = TRUE,
  forceLinear = TRUE,
  jackknife = FALSE,
  arguments = NULL,
  scratchDir = NULL,
  out = "model",
  cores = 1,
  verbose = FALSE,
  ...
)
```

Arguments

`data`	Data frame.
`resp`	Response variable. This is either the name of the column in `data` or an integer indicating the column in `data` that has the response variable. The default is to use the first column in `data` as the response.
`preds`	Character vector or integer vector. Names of columns or column indices of predictors. The default is to use the second and subsequent columns in `data`.

regMult	Numeric vector. Values of the master regularization parameters (called beta in some publications) to test.
classes	Character vector. Names of feature classes to use (either default to use lpqh) or any combination of lpqht, where l ==> linear features, p ==> product features, q ==> quadratic features, h ==> hinge features, and t ==> threshold features. Example: c('l', 'p', 'q').
testClasses	Logical. If TRUE (default) then test all possible combinations of classes (note that all tested models will at least have linear features). If FALSE then use the classes provided (these will not vary between models).
dropOverparam	Logical, if TRUE (default), drop models if they have more coefficients than training occurrences. It is possible for no models to fulfill this criterion, in which case no models will be returned.
anyway	Logical. Same as dropOverparam (included for backwards compatibility. If NULL (default), then the value of dropOverparam will take precedence. If TRUE or FALSE then anyway will override the value of dropOverparam.
forceLinear	Logical. If TRUE (default) then require any tested models to include at least linear features.
jackknife	Logical. If TRUE (default) the the returned model will be also include jackknife testing of variable importance.
arguments	NULL (default) or a character vector. Options to pass to maxent()'s args argument. (Do not include l, p, q, h, t, betamultiplier, or jackknife!)
scratchDir	Character. Directory to which to write temporary files. Leave as NULL to create a temporary folder in the current working directory.
out	Character vector. One or more values: • 'model': Model with the lowest AICc. • 'models': All models evaluated, sorted from lowest to highest AICc (lowest is best). • 'tuning': Data frame with tuning parameters, one row per model, sorted by AICc.
cores	Number of cores to use. Default is 1. If you have issues when cores > 1, please see the troubleshooting_parallel_operations guide.
verbose	Logical. If TRUE report progress and AICc table.
...	Extra arguments. Not used.

Details

The function can return the best model (default), a list of models created using all possible combinations of feature classes and regularization multipliers, and/or a data frame with tuning statistics for each model. Models in the list and in the data frame are sorted from best to worst. The function requires the maxent jar file. Its output is any or all of: a table with AICc for all evaluated models; all models evaluated in the "selection" phase; and/or the single model with the lowest AICc.

Note that due to differences in how MaxEnt and MaxNet are implemented in their base packages, the models will not necessarily be the same even for the same training data.

This function is a wrapper for `MaxEnt()`. The `MaxEnt` function creates a series of files on disk for each model. This function assumes you do not want those files, so deletes most of them. However, there is one that cannot be deleted and the normal ways of changing its permissions in **R** do not work. So the function simply writes over that file (which is allowed) to make it smaller. Regardless, if you run many models your temporary directory (argument `scratchDir`) can fill up and require manual deletion.

Value

The object that is returned depends on the value of the `out` argument. It can be a model object, a data frame, a list of models, or a list of all two or more of these.

References

Warren, D.L. and S.N. Siefert. 2011. Ecological niche modeling in Maxent: The importance of model complexity and the performance of model selection criteria. *Ecological Applications* 21:335-342. doi:10.1890/101171.1

See Also

`MaxEnt`

Examples

```
# NB: The examples below show a very basic modeling workflow. They have been
# designed to work fast, not produce accurate, defensible models. They can
# take a few minutes to run.

library(mgcv)
library(sf)
library(terra)
set.seed(123)

### setup data
###############

# environmental rasters
rastFile <- system.file('extdata/madClim.tif', package='enmSdmX')
madClim <- rast(rastFile)

# coordinate reference system
wgs84 <- getCRS('WGS84')

# lemur occurrence data
data(lemurs)
occs <- lemurs[lemurs$species == 'Eulemur fulvus', ]
occs <- vect(occs, geom=c('longitude', 'latitude'), crs=wgs84)

occs <- elimCellDuplicates(occs, madClim)

occEnv <- extract(madClim, occs, ID = FALSE)
```

```
occEnv <- occEnv[complete.cases(occEnv), ]

# create 10000 background sites (or as many as raster can support)
bgEnv <- terra::spatSample(madClim, 20000)
bgEnv <- bgEnv[complete.cases(bgEnv), ]
bgEnv <- bgEnv[1:min(10000, nrow(bgEnv)), ]

# collate occurrences and background sites
presBg <- data.frame(
  presBg = c(
    rep(1, nrow(occEnv)),
    rep(0, nrow(bgEnv))
  )
)

env <- rbind(occEnv, bgEnv)
env <- cbind(presBg, env)

predictors <- c('bio1', 'bio12')

### calibrate models
####################

# Note that all of the trainXYZ functions can made to go faster using the
# "cores" argument (set to just 1, by default). The examples below will not
# go too much faster using more cores because they are simplified, but
# you can try!
cores <- 1

# MaxEnt
mx <- trainMaxEnt(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)

# MaxNet
mn <- trainMaxNet(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)

# generalized linear model (GLM)
gl <- trainGLM(
 data = env,
 resp = 'presBg',
```

```
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 verbose = TRUE,
 cores = cores
)

# generalized additive model (GAM)
ga <- trainGAM(
 data = env,
 resp = 'presBg',
 preds = predictors,
 verbose = TRUE,
 cores = cores
)

# natural splines
ns <- trainNS(
 data = env,
 resp = 'presBg',
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 df = 1:2, # too few values for reliable model(?)
 verbose = TRUE,
 cores = cores
)

# boosted regression trees
envSub <- env[1:1049, ] # subsetting data to run faster
brt <- trainBRT(
 data = envSub,
 resp = 'presBg',
 preds = predictors,
 learningRate = 0.001, # too few values for reliable model(?)
 treeComplexity = c(2, 3), # too few values for reliable model, but fast
 minTrees = 1200, # minimum trees for reliable model(?), but fast
 maxTrees = 1200, # too small for reliable model(?), but fast
 tryBy = 'treeComplexity',
 anyway = TRUE, # return models that did not converge
 verbose = TRUE,
 cores = cores
)

# random forests
rf <- trainRF(
 data = env,
 resp = 'presBg',
 preds = predictors,
 numTrees = c(100, 500), # using at least 500 recommended, but fast!
 verbose = TRUE,
 cores = cores
)

### make maps of models
```

```
#######################

# NB We do not have to scale rasters before predicting GLMs and NSs because we
# used the `scale = TRUE` argument in trainGLM() and trainNS().

mxMap <- predictEnmSdm(mx, madClim)
mnMap <- predictEnmSdm(mn, madClim)
glMap <- predictEnmSdm(gl, madClim)
gaMap <- predictEnmSdm(ga, madClim)
nsMap <- predictEnmSdm(ns, madClim)
brtMap <- predictEnmSdm(brt, madClim)
rfMap <- predictEnmSdm(rf, madClim)

maps <- c(
 mxMap,
 mnMap,
 glMap,
 gaMap,
 nsMap,
 brtMap,
 rfMap
)

names(maps) <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NSs', 'BRTs', 'RFs')
fun <- function() plot(occs, col='black', pch=3, add=TRUE)
plot(maps, fun = fun, nc = 4)

### compare model responses to BIO12 (mean annual precipitation)
#####################################################################

# make a data frame holding all other variables at mean across occurrences,
# varying only BIO12
occEnvMeans <- colMeans(occEnv, na.rm=TRUE)
occEnvMeans <- rbind(occEnvMeans)
occEnvMeans <- as.data.frame(occEnvMeans)
climFrame <- occEnvMeans[rep(1, 100), ]
rownames(climFrame) <- NULL

minBio12 <- min(env$bio12)
maxBio12 <- max(env$bio12)
climFrame$bio12 <- seq(minBio12, maxBio12, length.out=100)

predMx <- predictEnmSdm(mx, climFrame)
predMn <- predictEnmSdm(mn, climFrame)
predGl <- predictEnmSdm(gl, climFrame)
predGa <- predictEnmSdm(ga, climFrame)
predNat <- predictEnmSdm(ns, climFrame)
predBrt <- predictEnmSdm(brt, climFrame)
predRf <- predictEnmSdm(rf, climFrame)


plot(climFrame$bio12, predMx,
xlab='BIO12', ylab='Prediction', type='l', ylim=c(0, 1))
```

```
lines(climFrame$bio12, predMn, lty='solid', col='red')
lines(climFrame$bio12, predGl, lty='dotted', col='blue')
lines(climFrame$bio12, predGa, lty='dashed', col='green')
lines(climFrame$bio12, predNat, lty=4, col='purple')
lines(climFrame$bio12, predBrt, lty=5, col='orange')
lines(climFrame$bio12, predRf, lty=6, col='cyan')

legend(
   'topleft',
   inset = 0.01,
   legend = c(
 'MaxEnt',
 'MaxNet',
 'GLM',
 'GAM',
 'NS',
 'BRT',
 'RF'
   ),
   lty = c(1, 1:6),
   col = c(
 'black',
 'red',
 'blue',
 'green',
 'purple',
 'orange',
 'cyan'
   ),
   bg = 'white'
)

### compare response curves ###
###############################

modelNames <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NS', 'BRT', 'RF')

responses <- responseCurves(
 models = list(mx, mn, gl, ga, ns, brt, rf),
 env = bgEnv,
 ref = occEnv,
 vars = predictors,
 modelNames = modelNames
)

print(responses)
```

`trainMaxNet` *Calibrate a MaxNet model using AICc*

Description

This function calculates the "best" MaxNet model using AICc across all possible combinations of a set of master regularization parameters and feature classes. The "best" model has the lowest AICc, with ties broken by number of features (fewer is better), regularization multiplier (higher better), then finally the number of coefficients (fewer better).

Usage

```
trainMaxNet(
  data,
  resp = names(data)[1],
  preds = names(data)[2:ncol(data)],
  regMult = c(seq(0.5, 5, by = 0.5), 7.5, 10),
  classes = "default",
  testClasses = TRUE,
  dropOverparam = TRUE,
  forceLinear = TRUE,
  out = "model",
  cores = 1,
  verbose = FALSE,
  ...
)
```

Arguments

`data`	Data frame or matrix. Contains a column indicating whether each row is a presence (1) or background (0) site, plus columns for environmental predictors.
`resp`	Character or integer. Name or column index of response variable. Default is to use the first column in `data`.
`preds`	Character vector or integer vector. Names of columns or column indices of predictors. Default is to use the second and subsequent columns in `data`.
`regMult`	Numeric vector. Values of the master regularization parameters (called `beta` in some publications) to test.
`classes`	Character vector. Names of feature classes to use (either `default` to use `'lpqh'`) or any combination of `'lpqht'`, where `l` ==> linear features, `p` ==> product features, `q` ==> quadratic features, `h` ==> hinge features, and `t` ==> threshold features. Example: `c('l', 'p', 'q')`.
`testClasses`	Logical. If `TRUE` (default) then test all possible combinations of classes (note that all tested models will at least have linear features). If `FALSE` then use the classes provided (these will not vary between models).

dropOverparam	Logical, if TRUE (default), drop models if they have more coefficients than training occurrences. It is possible for no models to fulfill this criterion, in which case no models will be returned.
forceLinear	Logical. If TRUE (default) then require any tested models to include at least linear features.
out	Character vector. One or more values: • 'model': Model with the lowest AICc. • 'models': All models evaluated, sorted from lowest to highest AICc (lowest is best). • 'tuning': Data frame with tuning parameters, one row per model, sorted by AICc.
cores	Number of cores to use. Default is 1. If you have issues when cores > 1, please see the troubleshooting_parallel_operations guide.
verbose	Logical. If TRUE report the AICc table.
...	Extra arguments. Not used.

Details

The function can return the best model (default), a list of models created using all possible combinations of feature classes and regularization multipliers, and/or a data frame with tuning statistics for each model. Models in the list and in the data frame are sorted from best to worst. Its output is any or all of: a table with AICc for all evaluated models; all models evaluated in the "selection" phase; and/or the single model with the lowest AICc.

Note that due to differences in how MaxEnt and MaxNet are implemented in their base packages, the models will not necessarily be the same even for the same training data.

Value

If out = 'model' this function returns an object of class MaxEnt. If out = 'tuning' this function returns a data frame with tuning parameters, log likelihood, and AICc for each model tried. If out = c('model', 'tuning' then it returns a list object with the MaxEnt object and the data frame.

References

Phillips, S.J., Anderson, R.P., Dudík, M. Schapire, R.E., and Blair, M.E. 2017. Opening the black box: An open-source release of Maxent. *Ecography* 40:887-893. doi:10.1111/ecog.03049 Warren, D.L. and S.N. Siefert. 2011. Ecological niche modeling in Maxent: The importance of model complexity and the performance of model selection criteria. *Ecological Applications* 21:335-342. doi:10.1890/101171.1

See Also

maxnet, MaxEnt, trainMaxEnt

Examples

```
# NB: The examples below show a very basic modeling workflow. They have been
# designed to work fast, not produce accurate, defensible models. They can
# take a few minutes to run.

library(mgcv)
library(sf)
library(terra)
set.seed(123)

### setup data
##############

# environmental rasters
rastFile <- system.file('extdata/madClim.tif', package='enmSdmX')
madClim <- rast(rastFile)

# coordinate reference system
wgs84 <- getCRS('WGS84')

# lemur occurrence data
data(lemurs)
occs <- lemurs[lemurs$species == 'Eulemur fulvus', ]
occs <- vect(occs, geom=c('longitude', 'latitude'), crs=wgs84)

occs <- elimCellDuplicates(occs, madClim)

occEnv <- extract(madClim, occs, ID = FALSE)
occEnv <- occEnv[complete.cases(occEnv), ]

# create 10000 background sites (or as many as raster can support)
bgEnv <- terra::spatSample(madClim, 20000)
bgEnv <- bgEnv[complete.cases(bgEnv), ]
bgEnv <- bgEnv[1:min(10000, nrow(bgEnv)), ]

# collate occurrences and background sites
presBg <- data.frame(
  presBg = c(
    rep(1, nrow(occEnv)),
    rep(0, nrow(bgEnv))
  )
)

env <- rbind(occEnv, bgEnv)
env <- cbind(presBg, env)

predictors <- c('bio1', 'bio12')

### calibrate models
####################
```

```
# Note that all of the trainXYZ functions can made to go faster using the
# "cores" argument (set to just 1, by default). The examples below will not
# go too much faster using more cores because they are simplified, but
# you can try!
cores <- 1

# MaxEnt
mx <- trainMaxEnt(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)

# MaxNet
mn <- trainMaxNet(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)

# generalized linear model (GLM)
gl <- trainGLM(
 data = env,
 resp = 'presBg',
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 verbose = TRUE,
 cores = cores
)

# generalized additive model (GAM)
ga <- trainGAM(
 data = env,
 resp = 'presBg',
 preds = predictors,
 verbose = TRUE,
 cores = cores
)

# natural splines
ns <- trainNS(
 data = env,
 resp = 'presBg',
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 df = 1:2, # too few values for reliable model(?)
 verbose = TRUE,
```

```
 cores = cores
)

# boosted regression trees
envSub <- env[1:1049, ] # subsetting data to run faster
brt <- trainBRT(
 data = envSub,
 resp = 'presBg',
 preds = predictors,
 learningRate = 0.001, # too few values for reliable model(?)
 treeComplexity = c(2, 3), # too few values for reliable model, but fast
 minTrees = 1200, # minimum trees for reliable model(?), but fast
 maxTrees = 1200, # too small for reliable model(?), but fast
 tryBy = 'treeComplexity',
 anyway = TRUE, # return models that did not converge
 verbose = TRUE,
 cores = cores
)

# random forests
rf <- trainRF(
 data = env,
 resp = 'presBg',
 preds = predictors,
 numTrees = c(100, 500), # using at least 500 recommended, but fast!
 verbose = TRUE,
 cores = cores
)

### make maps of models
#######################

# NB We do not have to scale rasters before predicting GLMs and NSs because we
# used the `scale = TRUE` argument in trainGLM() and trainNS().

mxMap <- predictEnmSdm(mx, madClim)
mnMap <- predictEnmSdm(mn, madClim)
glMap <- predictEnmSdm(gl, madClim)
gaMap <- predictEnmSdm(ga, madClim)
nsMap <- predictEnmSdm(ns, madClim)
brtMap <- predictEnmSdm(brt, madClim)
rfMap <- predictEnmSdm(rf, madClim)

maps <- c(
 mxMap,
 mnMap,
 glMap,
 gaMap,
 nsMap,
 brtMap,
 rfMap
)
```

```
names(maps) <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NSs', 'BRTs', 'RFs')
fun <- function() plot(occs, col='black', pch=3, add=TRUE)
plot(maps, fun = fun, nc = 4)

### compare model responses to BIO12 (mean annual precipitation)
#################################################################

# make a data frame holding all other variables at mean across occurrences,
# varying only BIO12
occEnvMeans <- colMeans(occEnv, na.rm=TRUE)
occEnvMeans <- rbind(occEnvMeans)
occEnvMeans <- as.data.frame(occEnvMeans)
climFrame <- occEnvMeans[rep(1, 100), ]
rownames(climFrame) <- NULL

minBio12 <- min(env$bio12)
maxBio12 <- max(env$bio12)
climFrame$bio12 <- seq(minBio12, maxBio12, length.out=100)

predMx <- predictEnmSdm(mx, climFrame)
predMn <- predictEnmSdm(mn, climFrame)
predGl <- predictEnmSdm(gl, climFrame)
predGa <- predictEnmSdm(ga, climFrame)
predNat <- predictEnmSdm(ns, climFrame)
predBrt <- predictEnmSdm(brt, climFrame)
predRf <- predictEnmSdm(rf, climFrame)


plot(climFrame$bio12, predMx,
xlab='BIO12', ylab='Prediction', type='l', ylim=c(0, 1))

lines(climFrame$bio12, predMn, lty='solid', col='red')
lines(climFrame$bio12, predGl, lty='dotted', col='blue')
lines(climFrame$bio12, predGa, lty='dashed', col='green')
lines(climFrame$bio12, predNat, lty=4, col='purple')
lines(climFrame$bio12, predBrt, lty=5, col='orange')
lines(climFrame$bio12, predRf, lty=6, col='cyan')

legend(
   'topleft',
   inset = 0.01,
   legend = c(
 'MaxEnt',
 'MaxNet',
 'GLM',
 'GAM',
 'NS',
 'BRT',
 'RF'
   ),
   lty = c(1, 1:6),
   col = c(
 'black',
```

```
 'red',
 'blue',
 'green',
 'purple',
 'orange',
 'cyan'
  ),
  bg = 'white'
)

### compare response curves ###
###############################

modelNames <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NS', 'BRT', 'RF')

responses <- responseCurves(
 models = list(mx, mn, gl, ga, ns, brt, rf),
 env = bgEnv,
 ref = occEnv,
 vars = predictors,
 modelNames = modelNames
)

print(responses)
```

---

trainNS *Calibrate a natural splines model*

Description

This function constructs a natural-spline model by evaluating all possible models given the available predictors and constraints. "Constraints" in this case include the degrees of freedom for a spline, whether or not interaction terms are included, minimum number of presence sites per model term, and maximum number of terms to include in the model. Its output is any or all of: the most parsimonious model (lowest AICc); all models evaluated; and/or a table with AICc for all evaluated models.

Usage

```
trainNS(
  data,
  resp = names(data)[1],
  preds = names(data)[2:ncol(data)],
  scale = NA,
  df = 1:4,
  interaction = TRUE,
```

```
  interceptOnly = TRUE,
  method = "glm.fit",
  presPerTermFinal = 10,
  maxTerms = 8,
  w = TRUE,
  family = "binomial",
  removeInvalid = TRUE,
  failIfNoValid = TRUE,
  out = "model",
  cores = 1,
  verbose = FALSE,
  ...
)
```

Arguments

data	Data frame.
resp	Response variable. This is either the name of the column in data or an integer indicating the column in data that has the response variable. The default is to use the first column in data as the response.
preds	Character vector or integer vector. Names of columns or column indices of predictors. The default is to use the second and subsequent columns in data.
scale	Either NA (default), or TRUE or FALSE. If TRUE, the predictors will be centered and scaled by dividing by subtracting their means then dividing by their standard deviations. The means and standard deviations will be returned in the model object under an element named "scales". For example, if you do something like model <- trainNS(data, scale=TRUE), then you can get the means and standard deviations using model$scales$means and model$scales$sds. If FALSE, no scaling is done. If NA (default), then the function will check to see if non-factor predictors have means ~0 and standard deviations ~1. If not, then a warning will be printed, but the function will continue to do its operations.
df	A vector of integers > 0 or NULL. Sets flexibility of model fit. See documentation for ns.
interaction	If TRUE (default), include two-way interaction terms.
interceptOnly	If TRUE (default) and model selection is enabled, then include an intercept-only model.
method	Character, name of function used to solve. This can be 'glm.fit' (default), 'brglmFit' (from the **brglm2** package), or another function.
presPerTermFinal	Minimum number of presence sites per term in initial starting model.
maxTerms	Maximum number of terms to be used in any model, not including the intercept (default is 8).
w	Weights. Any of: • TRUE: Causes the total weight of presences to equal the total weight of absences (if family='binomial')

- FALSE: Each datum is assigned a weight of 1.
- A numeric vector of weights, one per row in data.
- The name of the column in data that contains site weights.

family	Name of family for data error structure (see family).
removeInvalid	Logical. If TRUE (default), remove models that either did not converge or have parameter estimates near the boundaries (usually negative or positive infinity).
failIfNoValid	Logical. If TRUE (default), and the "full" model either does not converge or has parameters near the boundary, then the function will fail. If FALSE, then return NULL in this case.
out	Character vector. One or more values: • 'model': Model with the lowest AICc. • 'models': All models evaluated, sorted from lowest to highest AICc (lowest is best). • 'tuning': Data frame with tuning parameters, one row per model, sorted by AICc.
cores	Number of cores to use. Default is 1. If you have issues when cores > 1, please see the troubleshooting_parallel_operations guide.
verbose	Logical. If TRUE then display intermediate results on the display device. Default is FALSE.
...	Arguments to send to glm.

Details

This function is designed to find the most parsimonious model given the amount of calibration data that is available to it. ‘trainNS()‘ can work with any data, but has been designed to work specifically as a species distribution model where the response is either binary (default) or abundance. Specifically, it 1) identifies the most parsimonious model (lowest AICc) with 2) optimal flexibility (optimal degrees of freedom in splines) and 3) allows for (but does not require) interaction terms between predictors (if desired). If the defaults are used, the following procedure is applied:

- Constructing a set of simple model terms, each with 1 to 4 degrees of freedom. Terms can be univariate or bilabiate (two-way interactions). Predictors can be continuous or factors. If any simple models has convergence issues or boundary issues (coefficients that approach negative or positive infinity), it is removed.
- Constructing a series of models, each with one of the terms, then using the models to rank terms by AICc.
- From the top set of terms, creating a "full" model. The full model will ensure the maximum number of terms is <= ‘maxTerms‘, and that for each term, there are at least ‘presPerTermFinal‘ data points.
- All possible submodels, plus the full model, are evaluated and ranked by AICc. If a model has convergence or boundary issues, it is removed from the set. The most parsimonious model (lowest AICc) is returned.

Value

The object that is returned depends on the value of the `out` argument. It can be a model object, a data frame, a list of models, or a list of all two or more of these. If `scale` is `TRUE`, any model object will also have an element named `$scale`, which contains the means and standard deviations for predictors that are not factors. The data frame reports the AICc for all of the models evaluated, sorted by best to worst. The `converged` column indicates whether the model converged ("`TRUE`" is good), and the `boundary` column whether the model parameters are near the boundary (usually, negative or positive infinity; "`FALSE`" is good).

See Also

`ns`

Examples

```
# NB: The examples below show a very basic modeling workflow. They have been
# designed to work fast, not produce accurate, defensible models. They can
# take a few minutes to run.

library(mgcv)
library(sf)
library(terra)
set.seed(123)

### setup data
###############

# environmental rasters
rastFile <- system.file('extdata/madClim.tif', package='enmSdmX')
madClim <- rast(rastFile)

# coordinate reference system
wgs84 <- getCRS('WGS84')

# lemur occurrence data
data(lemurs)
occs <- lemurs[lemurs$species == 'Eulemur fulvus', ]
occs <- vect(occs, geom=c('longitude', 'latitude'), crs=wgs84)

occs <- elimCellDuplicates(occs, madClim)

occEnv <- extract(madClim, occs, ID = FALSE)
occEnv <- occEnv[complete.cases(occEnv), ]

# create 10000 background sites (or as many as raster can support)
bgEnv <- terra::spatSample(madClim, 20000)
bgEnv <- bgEnv[complete.cases(bgEnv), ]
bgEnv <- bgEnv[1:min(10000, nrow(bgEnv)), ]

# collate occurrences and background sites
```

```
presBg <- data.frame(
  presBg = c(
    rep(1, nrow(occEnv)),
    rep(0, nrow(bgEnv))
  )
)

env <- rbind(occEnv, bgEnv)
env <- cbind(presBg, env)

predictors <- c('bio1', 'bio12')

### calibrate models
####################

# Note that all of the trainXYZ functions can made to go faster using the
# "cores" argument (set to just 1, by default). The examples below will not
# go too much faster using more cores because they are simplified, but
# you can try!
cores <- 1

# MaxEnt
mx <- trainMaxEnt(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)

# MaxNet
mn <- trainMaxNet(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)

# generalized linear model (GLM)
gl <- trainGLM(
 data = env,
 resp = 'presBg',
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 verbose = TRUE,
 cores = cores
)

# generalized additive model (GAM)
ga <- trainGAM(
```

```
 data = env,
 resp = 'presBg',
 preds = predictors,
 verbose = TRUE,
 cores = cores
)

# natural splines
ns <- trainNS(
 data = env,
 resp = 'presBg',
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 df = 1:2, # too few values for reliable model(?)
 verbose = TRUE,
 cores = cores
)

# boosted regression trees
envSub <- env[1:1049, ] # subsetting data to run faster
brt <- trainBRT(
 data = envSub,
 resp = 'presBg',
 preds = predictors,
 learningRate = 0.001, # too few values for reliable model(?)
 treeComplexity = c(2, 3), # too few values for reliable model, but fast
 minTrees = 1200, # minimum trees for reliable model(?), but fast
 maxTrees = 1200, # too small for reliable model(?), but fast
 tryBy = 'treeComplexity',
 anyway = TRUE, # return models that did not converge
 verbose = TRUE,
 cores = cores
)

# random forests
rf <- trainRF(
 data = env,
 resp = 'presBg',
 preds = predictors,
 numTrees = c(100, 500), # using at least 500 recommended, but fast!
 verbose = TRUE,
 cores = cores
)

### make maps of models
#######################

# NB We do not have to scale rasters before predicting GLMs and NSs because we
# used the `scale = TRUE` argument in trainGLM() and trainNS().

mxMap <- predictEnmSdm(mx, madClim)
mnMap <- predictEnmSdm(mn, madClim)
glMap <- predictEnmSdm(gl, madClim)
```

```
gaMap <- predictEnmSdm(ga, madClim)
nsMap <- predictEnmSdm(ns, madClim)
brtMap <- predictEnmSdm(brt, madClim)
rfMap <- predictEnmSdm(rf, madClim)

maps <- c(
 mxMap,
 mnMap,
 glMap,
 gaMap,
 nsMap,
 brtMap,
 rfMap
)

names(maps) <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NSs', 'BRTs', 'RFs')
fun <- function() plot(occs, col='black', pch=3, add=TRUE)
plot(maps, fun = fun, nc = 4)

### compare model responses to BIO12 (mean annual precipitation)
#####################################################################

# make a data frame holding all other variables at mean across occurrences,
# varying only BIO12
occEnvMeans <- colMeans(occEnv, na.rm=TRUE)
occEnvMeans <- rbind(occEnvMeans)
occEnvMeans <- as.data.frame(occEnvMeans)
climFrame <- occEnvMeans[rep(1, 100), ]
rownames(climFrame) <- NULL

minBio12 <- min(env$bio12)
maxBio12 <- max(env$bio12)
climFrame$bio12 <- seq(minBio12, maxBio12, length.out=100)

predMx <- predictEnmSdm(mx, climFrame)
predMn <- predictEnmSdm(mn, climFrame)
predGl <- predictEnmSdm(gl, climFrame)
predGa <- predictEnmSdm(ga, climFrame)
predNat <- predictEnmSdm(ns, climFrame)
predBrt <- predictEnmSdm(brt, climFrame)
predRf <- predictEnmSdm(rf, climFrame)


plot(climFrame$bio12, predMx,
xlab='BIO12', ylab='Prediction', type='l', ylim=c(0, 1))

lines(climFrame$bio12, predMn, lty='solid', col='red')
lines(climFrame$bio12, predGl, lty='dotted', col='blue')
lines(climFrame$bio12, predGa, lty='dashed', col='green')
lines(climFrame$bio12, predNat, lty=4, col='purple')
lines(climFrame$bio12, predBrt, lty=5, col='orange')
lines(climFrame$bio12, predRf, lty=6, col='cyan')
```

```
legend(
   'topleft',
   inset = 0.01,
   legend = c(
 'MaxEnt',
 'MaxNet',
 'GLM',
 'GAM',
 'NS',
 'BRT',
 'RF'
   ),
   lty = c(1, 1:6),
   col = c(
 'black',
 'red',
 'blue',
 'green',
 'purple',
 'orange',
 'cyan'
   ),
   bg = 'white'
)

### compare response curves ###
###############################

modelNames <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NS', 'BRT', 'RF')

responses <- responseCurves(
 models = list(mx, mn, gl, ga, ns, brt, rf),
 env = bgEnv,
 ref = occEnv,
 vars = predictors,
 modelNames = modelNames
)

print(responses)
```

`trainRF` *Calibrate a random forest model*

Description

This function trains a random forest model. It identifies the optimal number of trees and value for `mtry` (number of variables sampled as candidates at each split) using out-of-bag error (OOB).

The number of trees in each candidate model is set by the user with argument `numTrees`. The number of predictors to test per split, `mtry`, is found by exploring a range of values. If the response (`y`) is a factor, the starting value for `mtry` is `max(1, floor(p / 3))`, where `p` is the number of predictors. If the response is not a factor, the starting value is `max(1, floor(sqrt(p)))`. Values `ymtryIncrement` argument until the total number of predictors is used. See `ranger` for more details.

The output of the function is any or all of: a table with out-of-bag (OOB) error of evaluated models; all evaluated models; and/or the single model with the lowest OOB error.

Usage

```
trainRF(
  data,
  resp = names(data)[1],
  preds = names(data)[2:ncol(data)],
  numTrees = c(250, 500, 750, 1000),
  mtryIncrement = 2,
  w = TRUE,
  binary = TRUE,
  out = "model",
  cores = 1,
  verbose = FALSE,
  ...
)
```

Arguments

`data`	Data frame.
`resp`	Response variable. This is either the name of the column in `data` or an integer indicating the column in `data` that has the response variable. The default is to use the first column in `data` as the response.
`preds`	Character vector or integer vector. Names of columns or column indices of predictors. The default is to use the second and subsequent columns in `data`.
`numTrees`	Vector of number of trees to grow. All possible combinations of `mtry` and `numTrees` will be assessed.
`mtryIncrement`	Positive integer (default is 2). Number of predictors to add to `mtry` until all predictors are in each tree.
`w`	Weights. For random forests, weights are simply used as relative probabilities of selecting a row in `data` to be used in a particular tree. This argument takes any of: • `TRUE`: Causes the total weight of presences to equal the total weight of absences (if `binary = TRUE`) • `FALSE`: Each datum is assigned a weight of 1. • A numeric vector of weights, one per row in `data`. • The name of the column in `data` that contains site weights.
`binary`	Logical. If `TRUE` (default) then the response is converted to a binary factor with levels 0 and 1. Otherwise, this argument has no effect and the response will be assumed to be a real number.

out	Character vector. One or more values: • 'model': Model with the lowest out-of-bag (OOB) error rate. • 'models': All models evaluated, sorted from lowest to highest OOB. • 'tuning': Data frame with tuning parameters, one row per model, sorted by OOB error rate.
cores	Number of cores to use. Default is 1. If you have issues when cores > 1, please see the troubleshooting_parallel_operations guide.
verbose	Logical. If TRUE then display progress for finding optimal value of mtry.
...	Arguments to pass to ranger. Of note, num.threads will allow for multi-threaded computation of each RF. However, it could be problemmatic to use this when cores > 1. Also of note, save.memory reduces speed but may make larger jobs possible.

Value

The object that is returned depends on the value of the out argument. It can be a model object, a data frame, a list of models, or a list of all two or more of these.

See Also

ranger

Examples

```
# NB: The examples below show a very basic modeling workflow. They have been
# designed to work fast, not produce accurate, defensible models. They can
# take a few minutes to run.

library(mgcv)
library(sf)
library(terra)
set.seed(123)

### setup data
##############

# environmental rasters
rastFile <- system.file('extdata/madClim.tif', package='enmSdmX')
madClim <- rast(rastFile)

# coordinate reference system
wgs84 <- getCRS('WGS84')

# lemur occurrence data
data(lemurs)
occs <- lemurs[lemurs$species == 'Eulemur fulvus', ]
occs <- vect(occs, geom=c('longitude', 'latitude'), crs=wgs84)

occs <- elimCellDuplicates(occs, madClim)
```

```
occEnv <- extract(madClim, occs, ID = FALSE)
occEnv <- occEnv[complete.cases(occEnv), ]

# create 10000 background sites (or as many as raster can support)
bgEnv <- terra::spatSample(madClim, 20000)
bgEnv <- bgEnv[complete.cases(bgEnv), ]
bgEnv <- bgEnv[1:min(10000, nrow(bgEnv)), ]

# collate occurrences and background sites
presBg <- data.frame(
  presBg = c(
    rep(1, nrow(occEnv)),
    rep(0, nrow(bgEnv))
  )
)

env <- rbind(occEnv, bgEnv)
env <- cbind(presBg, env)

predictors <- c('bio1', 'bio12')

### calibrate models
####################

# Note that all of the trainXYZ functions can made to go faster using the
# "cores" argument (set to just 1, by default). The examples below will not
# go too much faster using more cores because they are simplified, but
# you can try!
cores <- 1

# MaxEnt
mx <- trainMaxEnt(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)

# MaxNet
mn <- trainMaxNet(
 data = env,
 resp = 'presBg',
 preds = predictors,
 regMult = 1, # too few values for reliable model, but fast
 verbose = TRUE,
 cores = cores
)

# generalized linear model (GLM)
gl <- trainGLM(
```

```
 data = env,
 resp = 'presBg',
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 verbose = TRUE,
 cores = cores
)

# generalized additive model (GAM)
ga <- trainGAM(
 data = env,
 resp = 'presBg',
 preds = predictors,
 verbose = TRUE,
 cores = cores
)

# natural splines
ns <- trainNS(
 data = env,
 resp = 'presBg',
 preds = predictors,
 scale = TRUE, # automatic scaling of predictors
 df = 1:2, # too few values for reliable model(?)
 verbose = TRUE,
 cores = cores
)

# boosted regression trees
envSub <- env[1:1049, ] # subsetting data to run faster
brt <- trainBRT(
 data = envSub,
 resp = 'presBg',
 preds = predictors,
 learningRate = 0.001, # too few values for reliable model(?)
 treeComplexity = c(2, 3), # too few values for reliable model, but fast
 minTrees = 1200, # minimum trees for reliable model(?), but fast
 maxTrees = 1200, # too small for reliable model(?), but fast
 tryBy = 'treeComplexity',
 anyway = TRUE, # return models that did not converge
 verbose = TRUE,
 cores = cores
)

# random forests
rf <- trainRF(
 data = env,
 resp = 'presBg',
 preds = predictors,
 numTrees = c(100, 500), # using at least 500 recommended, but fast!
 verbose = TRUE,
 cores = cores
)
```

```
### make maps of models
#######################

# NB We do not have to scale rasters before predicting GLMs and NSs because we
# used the `scale = TRUE` argument in trainGLM() and trainNS().

mxMap <- predictEnmSdm(mx, madClim)
mnMap <- predictEnmSdm(mn, madClim)
glMap <- predictEnmSdm(gl, madClim)
gaMap <- predictEnmSdm(ga, madClim)
nsMap <- predictEnmSdm(ns, madClim)
brtMap <- predictEnmSdm(brt, madClim)
rfMap <- predictEnmSdm(rf, madClim)

maps <- c(
 mxMap,
 mnMap,
 glMap,
 gaMap,
 nsMap,
 brtMap,
 rfMap
)

names(maps) <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NSs', 'BRTs', 'RFs')
fun <- function() plot(occs, col='black', pch=3, add=TRUE)
plot(maps, fun = fun, nc = 4)

### compare model responses to BIO12 (mean annual precipitation)
################################################################

# make a data frame holding all other variables at mean across occurrences,
# varying only BIO12
occEnvMeans <- colMeans(occEnv, na.rm=TRUE)
occEnvMeans <- rbind(occEnvMeans)
occEnvMeans <- as.data.frame(occEnvMeans)
climFrame <- occEnvMeans[rep(1, 100), ]
rownames(climFrame) <- NULL

minBio12 <- min(env$bio12)
maxBio12 <- max(env$bio12)
climFrame$bio12 <- seq(minBio12, maxBio12, length.out=100)

predMx <- predictEnmSdm(mx, climFrame)
predMn <- predictEnmSdm(mn, climFrame)
predGl <- predictEnmSdm(gl, climFrame)
predGa <- predictEnmSdm(ga, climFrame)
predNat <- predictEnmSdm(ns, climFrame)
predBrt <- predictEnmSdm(brt, climFrame)
predRf <- predictEnmSdm(rf, climFrame)
```

```
plot(climFrame$bio12, predMx,
xlab='BIO12', ylab='Prediction', type='l', ylim=c(0, 1))

lines(climFrame$bio12, predMn, lty='solid', col='red')
lines(climFrame$bio12, predGl, lty='dotted', col='blue')
lines(climFrame$bio12, predGa, lty='dashed', col='green')
lines(climFrame$bio12, predNat, lty=4, col='purple')
lines(climFrame$bio12, predBrt, lty=5, col='orange')
lines(climFrame$bio12, predRf, lty=6, col='cyan')

legend(
   'topleft',
   inset = 0.01,
   legend = c(
 'MaxEnt',
 'MaxNet',
 'GLM',
 'GAM',
 'NS',
 'BRT',
 'RF'
   ),
   lty = c(1, 1:6),
   col = c(
 'black',
 'red',
 'blue',
 'green',
 'purple',
 'orange',
 'cyan'
   ),
   bg = 'white'
)

### compare response curves ###
###############################

modelNames <- c('MaxEnt', 'MaxNet', 'GLM', 'GAM', 'NS', 'BRT', 'RF')

responses <- responseCurves(
 models = list(mx, mn, gl, ga, ns, brt, rf),
 env = bgEnv,
 ref = occEnv,
 vars = predictors,
 modelNames = modelNames
)

print(responses)
```

troubleshooting_parallel_operations
Troubleshooting parallel operations

Description

This is a guide to solving issues with running functions that can use more than one core. This includes the `train`*XYZ* functions, `bioticVelocity`, and `predictEnmSdm`. Each of these function has the argument `cores`. By default, the value of `cores` is 1, so the function will use only one core. By setting this higher, you can use more cores on your machine. However, occasionally you will run into the error:
`Error in checkForRemoteErrors(lapply(cl, recvResult)) :`
`2 nodes produced errors; first error: object '.doSnowGlobals' not found`
This means that the worker "nodes" (different instances of R started by the function to run in parallel) cannot find the **doParallel** package, even if it is installed on your system.

There are several solutions to this issue. One of them may work for you, and none are inherent to **enmSdmX**, as far as I can tell.

Anti-virus is blocking R

Strangely enough, running R in parallel sometimes looks like you are accessing the internet to anti-virus software. So, it may block access to other instances of R. You will have to do some surgery on your anti-virus software settings to find where to change this.

Your R packages are not stored in the "traditional" place

R has a default directory where packages are stored on any system. If your packages are stored in a different place, worker nodes may not be able to find them *if* you use `setwd` to change the working directory. I do not know if you have to set the working directory back to the default for your system, or if you have to change it to the folder that *contains* the folder where your R packages reside (for me, they are the same directory). You can see what your current working directory is using `getwd`. RStudio will often change this directory automatically.

So, if you get this error, try using `setwd` to set your working directory to the default one for your system, or to the folder that contains the folder that contains your packages.

Let me know

I'm always game to help you track down your problems (with this package, not necessarily in general). The best way is to create an issue on GitHub.

Exorcise your computer

Not responsible for damage to your computer.

`weightByDist` *Proximity-based weighting for occurrences to correct for spatial bias*

Description

This function calculates weights for points based on proximity to other points and the distance of spatial autocorrelation.

Usage

```
weightByDist(x, maxDist, alpha = 1)
```

Arguments

`x`	A spatial points object of class `SpatVector` or `sf`.
`maxDist`	Maximum distance beyond which a two neighboring points are assumed to have no effect on one another for calculation of weights.
`alpha`	Scaling parameter (see equations below).

Details

Weights can be used, for example, to account for spatial bias in the manner in which the points were observed. Weighting is calculated on the assumption that if two points fell exactly on top of one another, they should each have a weight of 1/2. If three points had the exact same coordinates, then their weights should be 1/3, and so on. Increasing distance between points should increase their weight, up to the distance at which there is no "significant" spatial autocorrelation, beyond which a point should have a weight of 1. This distance needs to be supplied by the user, as it will depend on the intended use of the weights. The distance can be calculated from "standard" metrics of spatial autocorrelation (e.g., a variogram), or on the basis of knowledge of the system (e.g., maximum dispersal distance of an organism).

For a given point i, the weight is defined as

$$w_i = 1/(1 + \epsilon)$$

where

$$\epsilon = \sum_{n=1}^{N}((1 - d_n)/d_sac)^{\alpha}$$

in which N is the total number of points closer than the maximum distance (d_sac) of point i, and d_n the distance between focal point i and point n. α is a weighting factor. By default, this is set to 1, but can be changed by the user to augment or diminish the effect that neighboring points have on the weight of a focal cell. When α is <1, neighboring points will reduce the weight of the focal point relative to the default, and when α is >1, they will have less effect relative to the default. When all neighboring points are at or beyond the maximum distance of spatial autocorrelation, then the focal point gets a weight w_i of 1. When at least neighboring one point is less than this distance away, the weight of the focal point will be >0 but <1.

Value

A numeric vector of weights.

Examples

```
library(sf)

# lemur occurrence data
data(lemurs)
wgs84 <- getCRS('WGS84')
occs <- lemurs[lemurs$species == 'Eulemur fulvus', ]
occs <- sf::st_as_sf(occs, coords=c('longitude', 'latitude'), crs=wgs84)

# weights
maxDist <- 30000 # in meters, for this example
w <- weightByDist(occs, maxDist)

# plot
plot(st_geometry(occs), cex=5 * w, main='point size ~ weight')
plot(st_geometry(mad0), col='gainsboro', border='gray70', add=TRUE)
plot(st_geometry(occs), cex=5 * w, add=TRUE)
```

Index

∗ **CRS**
  crss, 21
∗ **Canada**
  canada, 15
∗ **Madagascar**
  lemurs, 52
  mad0, 53
  mad1, 54
  madClim, 55
  madClim2030, 55
  madClim2050, 56
  madClim2070, 57
  madClim2090, 57
∗ **climate**
  madClim, 55
  madClim2030, 55
  madClim2050, 56
  madClim2070, 57
  madClim2090, 57
∗ **coordinate**
  crss, 21
∗ **lemurs**
  lemurs, 52
∗ **projection**
  crss, 21
∗ **tutorial**
  troubleshooting_parallel_operations, 161
.libPaths, *6*

approxfun, *50*
approximate, *50*

bioticVelocity, 3, *161*
bs, *50*

canada, 15
compareResponse, 16, *70*
coordImprecision, 18
cor, *29*
countPoints, 20
crss, 21
customAlbers, 22, *22*
customLambert, *22*
customLambert *(*customAlbers*)*, 22
customVNS, *22*
customVNS *(*customAlbers*)*, 22

decimalToDms, 24
dmsToDecimal, 24

elimCellDuplicates, 25
evalAUC, 26, *29–31*, *33*, *35, 36*, *38*, *98*, *111*
evalContBoyce, 28, *29*, *31*, *33*, *35, 36*, *38*, *98*, *111*
evalMultiAUC, *27*, *29*, 30, *33*, *35, 36*, *38*
evalThreshold, *29*, 31, *31*, *35, 36*, *38*
evalThresholdStats, *29*, *31*, 33, *33*, *36*, *38*
evalTjursR2, *29*, *31*, *33*, 35, *35*, *38*
evalTSS, *29*, *31*, *33*, *35, 36*, 37, *98*, *112*
extentToVect, 38, *70*

family, *117*, *128*, *149*

gam, *49, 50*, *120, 121*
gbm, *103*, *105*
geoFold, 39, *42*
geoFoldContrast, *40*, 41
geoThin, 43
getCRS, *21, 22*, 45
getValueByCell, 46
getwd, *161*
glm, *49, 50*, *117*, *129*, *149*
global, *47, 48*
globalx, 47

hclust, *40*, *44*

interpolateRasts, 48

lemurs, 52

longLatRasts, 52

mad0, 53
mad1, 54
madClim, 55
madClim2030, 55
madClim2050, 56
madClim2070, 57
madClim2090, 57
MaxEnt, *78*, *136*, *142*
maxnet, *84*, *142*
modelSize, 58

nearestEnvPoints, 59, *65*
nearestGeogPoints, *61*, 63
nicheOverlapMetrics, *5*, *18*, 69
ns, *148*, *150*

pa_evaluate, *27*, *29*, *31*, *33*, *35, 36*, *38*
plot_grid, *90*
plotExtent, *39*, 70
prcomp, *60*
predict, *72*, *77*, *84*
predictEnmSdm, 71, *89*, *161*
predictMaxEnt, *71, 72*, 77
predictMaxNet, *72*, 83

ranger, *99*, *155, 156*
responseCurves, 89

s, *120*
sampleRast, 95
setValueByCell *(*getValueByCell*)*, 46
setValues, *47*
setwd, *161*
smooth.spline, *50*
smooth.terms, *120*
spatSample, *95*
spatVectorToSpatial, 96
splinefun, *50*
squareCellRast, 97
summaryByCrossValid, 98, *112*

threshold, *32, 33*, *35*
trainBRT, *99*, 103, *111, 112*, *117*
trainByCrossValid, *98, 99*, 109, *117*
trainESM, 115
trainGAM, *99*, *111, 112*, *117*, 119
trainGLM, *72*, *99*, *111, 112*, *117*, 126
trainMaxEnt, *99*, *111, 112*, *117*, 134, *142*
trainMaxNet, *112*, *117*, 141
trainNS, *49, 50*, *72*, *99*, *111, 112*, *117*, 147
trainRF, *99*, *111, 112*, *117*, 154
troubleshooting_parallel_operations, *6*, *72*, *104*, *121*, *128*, *135*, *142*, *149*, *156*, 161

values, *47*

weightByDist, 162