Package ‘SAiVE’

April 22, 2024

Type Package

Title Functions Used for SAiVE Group Research, Collaborations, and Publications

Version 1.0.5

Date/Publication 2024-04-22 08:22:53 UTC

Description Holds functions developed by the University of Ottawa's SAiVE (Spatio-temporal Analysis of isotope Variations in the Environment) research group with the intention of facilitating the re-use of code, foster good code writing practices, and to allow others to benefit from the work done by the SAiVE group. Contributions are welcome via the 'GitHub' repository <https://github.com/UO-SAiVE/SAiVE> by group members as well as non-members.

License MIT + file LICENSE

URL https://github.com/UO-SAiVE/SAiVE

BugReports https://github.com/UO-SAiVE/SAiVE/issues

Depends R (>= 4.2)

Imports caret, crayon, doParallel, parallel, proxy, rlang, stats, terra, utils, VSURF

Suggests ranger, testthat (>= 3.0.0), vdiffr, whitebox

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

NeedsCompilation no

Author Ghislain de Laplante [aut, cre, cph] (<https://orcid.org/0000-0002-5093-9185>)

Maintainer Ghislain de Laplante <ghislain.delaplante@yukon.ca>

Repository CRAN

R topics documented:

`aspect` *Raster of aspect*

Description

Raster of aspect

Usage

```
aspect
```

Format

`aspect`:
A tif file loaded as a terra spatRaster

Source

Derived from the Canadian Digital Elevation Model DEM

`basin_dem` *Raster of elevation for basin delineation testing*

Description

Raster of elevation for basin delineation testing

Usage

```
basin_dem
```

Format

`basin_dem`**:**
A tif file loaded as a terra spatRaster

Source

Small subset of the Canadian Digital Elevation Model DEM

`basin_pts` *Points for basin delineation*

Description

Points for basin delineation

Usage

```
basin_pts
```

Format

`basin_pts`**:**
A geopackage file loaded as a terra spatVector

Source

Created by package developer in ArcGIS Pro `createStreams` *Create stream network from DEM*

Description

[Stable]

Creates a stream network from a provided DEM. In most cases it is advisable to first hydro-process the DEM (see `hydroProcess()`) to remove depressions which preclude continuous flow from one DEM cell to the next.

Usage

```
createStreams(
  DEM,
  threshold,
  vector = NULL,
  save_path = NULL,
  force_update_wbt = FALSE,
  n.cores = NULL
)
```

Arguments

`DEM`	The path to a digital elevation model file with .tif extension, or a terra spatRaster object. It is usually advisable to have already hydro-processed the DEM to remove artificial depressions. See `hydroProcess()`.
`threshold`	The accumulation threshold in DEM cells necessary to start defining a stream.
`vector`	Output file specifications. NULL for no vector file saved to disk, "gpkg" for a geopackage file, "shp" for a shapefile.
`save_path`	An optional path in which to save the newly created stream network. If left NULL will save it in the same directory as the provided DEM or, if the DEM is a terra object, return only terra objects.
`force_update_wbt`	Whitebox Tools is by default only downloaded if it cannot be found on the computer, and no check are performed to ensure the local version is current. Set to TRUE if you know that there is a new version and you would like to use it.
`n.cores`	The maximum number of cores to use. Leave NULL to use all cores minus 1.

Details

This function is essentially a convenient wrapper around three WhiteboxTools geospatial tools: `whitebox::wbt_d8_flow_accumulation()`, `whitebox::wbt_d8_pointer()`, and `whitebox::wbt_extract_streams()`

Value

A raster representation of streams and, if requested, a vector representation of streams. Returned as terra objects and saved to disk if `save_path` is not null.

Author(s)

Ghislain de Laplante (gdela069@uottawa.ca or ghislain.delaplante@yukon.ca)

Examples

```
hydroDEM <- hydroProcess(elev, 200, streams, n.cores = 2)
res <- createStreams(hydroDEM, 50, n.cores = 2)

terra::plot(res$streams_derived)
```

`drainageBasins` *Watershed/basin delineation*

Description

[Stable]

Hydro-processes a DEM, creating flow accumulation, direction, and streams rasters, and (optionally) delineates watersheds above one or more points using Whitebox Tools. To facilitate this task in areas with poor quality/low resolution DEMs, can "burn-in" a stream network to the DEM to ensure proper stream placement (see details). Many time-consuming raster operations are performed, so the function will attempt to use existing rasters if they are present in the same path as the base DEM and named according to the function's naming conventions. In practice, this means that only the first run of the function needs to be very time consuming. See details for more information.

NOTE 1: This tool can be slow to execute, and will use a lot of memory. Be patient, it might take several hours with a large DEM.

NOTE 2: ESRI shapefiles, on which the Whitebox Tools functions depend, truncate column names to 10 characters. You may want to save and re-assign column names to the output terra object after this function has run.

NOTE 3: If you are have already run this tool and are using a DEM in the same directory as last time, you only need to specify the DEM and the points (and, optionally, a projection for the points output). Operations using the optional streams shapefile and generating flow accumulation direction, and the artificial streams raster do not need to be repeated unless you want to use a different DEM or streams shapefile.

NOTE 4: This function is very memory (RAM) intensive. You'll want at least 16GB of RAM, and to ensure that most of it is free. If you get an error such as 'cannot allocate xxxxx bytes', you probably don't have the resources to run the tool. All rasters are un-compressed and converted to 64-bit float type before starting work, and there needs to be room to store more than twice that uncompressed raster size in memory.

Usage

```
drainageBasins(
  save_path,
  DEM,
  streams = NULL,
  breach_dist = 10000,
  threshold = 500,
  overwrite = FALSE,
  points = NULL,
  points_name_col = NULL,
  projection = NULL,
  snap = "nearest",
  snap_dist = 200,
  force_update_wbt = FALSE,
  n.cores = NULL
)
```

Arguments

`save_path`	The directory where you want the output shapefiles saved.
`DEM`	The path to a DEM including extension from which to delineate watersheds/catchments. Must be in .tif format. Derived layers such as flow accumulation, flow direction, and streams will inherit the DEM coordinate reference system.
`streams`	Optionally, the path to the polylines shapefile/geopackage containing lines, which can be used to improve accuracy when using poor quality DEMs. If this shapefile is the only input parameter being modified from previous runs (i.e. you've found a new/better streams shapefile but the DEM is unchanged) then specify a shapefile or geopackage lines file here and overwrite = TRUE.
`breach_dist`	The max radius (in raster cells) for which to search for a path to breach depressions, passed to `whitebox::wbt_breach_depressions_least_cost()`. This value should be high to ensure all depressions are breached. Note that the DEM is *not* breached in order of lowest elevation to greatest, nor is it breached sequentially (order is unknown, but the raster is presumably searched in some grid pattern for depressions). This means that flow paths may need to cross multiple depressions, especially in low relief areas.
`threshold`	The accumulation threshold in DEM cells necessary to start defining a stream. This streams raster is necessary to snap pout points to, so make sure not to make this number too great!
`overwrite`	If applicable, should rasters present in the same directory as the DEM be overwritten? This will also force the recalculation of derived layers.
`points`	The path to the points shapefile (extension .shp) containing the points from which to build watersheds. The attribute of each point will be attached to the newly-created drainage polygons. Leave NULL (along with related parameters) to only process the DEM without defining watersheds.
`points_name_col`	The name of the column in the points shapefile containing names to assign to the

	watersheds. Duplicates *are* allowed, and are labelled with the suffix _duplicate and a number for duplicates 2+.
`projection`	Optionally, a projection string in the form "epsg:3579" (find them here). The derived watersheds and point output layers will use this projection. If NULL the projection of the points will be used.
`snap`	Snap to the "nearest" derived (calculated) stream, or to the "greatest" flow accumulation cell within the snap distance? Beware that "greatest" will move the point downstream by up to the 'snap_dist' specified, while nearest might snap to the wrong stream.
`snap_dist`	The search radius within which to snap points to streams. Snapping method depends on 'snap' parameter. Note that distance units will match the projection, so probably best to work on a meter grid.
`force_update_wbt`	Whitebox Tools is by default only downloaded if it cannot be found on the computer, and no check are performed to ensure the local version is current. Set to TRUE if you know that there is a new version and you would like to use it.
`n.cores`	The maximum number of cores to use. Leave NULL to use all cores minus 1.

Details

This function uses software from the Whitebox geospatial analysis package, built by Prof. John Lindsay. Refer to this link for more information.

Creating derived raster layers without defining watersheds:

This function can be run without having any specific point above which to define a watershed. This can come in handy if you need to know where the synthetic streams raster will end up to ensure that your defined watershed pour points do not end up on the wrong stream branch, or if you simply want to front-load work while you work on defining the watershed pour points. To do this, leave the parameter 'points' and associated parameters as `NULL`.

Explanation of process::

Starting from a supplied DEM, the function will fill single-cell pits, burn-in a stream network depression if requested (ensuring that flow accumulations happen in the correct location), breach depressions in the digital elevation model using a least-cost algorithm (i.e. using the pathway resulting in minimal changes to the DEM considering distance and elevation) then calculate flow accumulation and direction rasters. Then, a raster of streams is created where flow accumulation is greatest. The points provided by the user are then snapped to the derived streams raster and watersheds are computed using the flow direction rasters. Finally, the watershed/drainage basin polygons are saved to the specified save path along with the provided points and the snapped pour points.

Using a streams shapefile to burn-in depressions to the DEM::

Be aware that this part of the function should ideally be used with a "simplified" streams shapefile. In particular, avoid or pre-process stream shapefiles that represent side-channels, as these will burn-in several parallel tracks to the DEM. ESRI has a tool called "simplify hydrology lines" which is great if you can ever get it to work, and WhiteboxTools has functions `whitebox::wbt_remove_short_streams()` to trim the streams raster, and `whitebox::wbt_repair_stream_vector_topology()` to help in converting a corrected streams vector to raster in the first place.

Value

A list of terra objects. If points are specified: delineated drainages, pour points as provided, snapped pour points, and the derived streams network. If no points: flow accumulation and direction rasters, and the derived streams network. If points specified, also saved to disk: an ESRI shapefile for each drainage basin, plus the associated snapped pour point and the point as provided and a shapefiles for all basins/points together. In all cases the created or discovered rasters will be in the same folder as the DEM.

Author(s)

Ghislain de Laplante (gdela069@uottawa.ca or ghislain.delaplante@yukon.ca)

Examples

```
# Must be run with file paths as well as a save_path

# Interim raster are created in the same path as the DEM

file.copy(system.file("extdata/basin_rast.tif", package = "SAiVE"),
  paste0(tempdir(), "/basin_rast.tif"))

basins <- drainageBasins(save_path = tempdir(),
  DEM = paste0(tempdir(), "/basin_rast.tif"),
  streams = system.file("extdata/streams.gpkg", package = "SAiVE"),
  points = system.file("extdata/basin_pts.gpkg", package = "SAiVE"),
  points_name_col = "ID",
  breach_dist = 500,
  n.cores = 2)

terra::plot(basins$delineated_basins)
```

`elev` *Raster of elevation*

Description

Raster of elevation

Usage

```
elev
```

Format

`elev`:
A tif file loaded as a terra spatRaster

Source

Small subset of the Canadian Digital Elevation Model DEM

`hydroProcess` *Hydro-process a DEM*

Description

[Stable]

Takes a digital elevation model and prepares it for hydrological analyses, such as basin delineation. Modifies the input DEM by breaching single cell pits/depressions and then breaching remaining depressions using a least cost algorithm (where cost is a function of distance plus elevation change to the DEM).

If a streams layer is specified, a depression will be "burned-in" to the DEM along the stream path (after converting the vector file to a raster). This is very useful when trying to delineate basins with a poor resolution DEM. You can control the depth of this depression with parameter 'burn_dist'.

Usage

```
hydroProcess(
  DEM,
  breach_dist,
  streams = NULL,
  burn_dist = 10,
  save_path = NULL,
  n.cores = NULL,
  force_update_wbt = FALSE
)
```

Arguments

`DEM`	The path to a digital elevation raster file with .tif extension, or a terra spatRaster object.
`breach_dist`	The max radius (in raster cells) in which to search for a path to breach depressions, passed to `whitebox::wbt_breach_depressions_least_cost()`. This value should be high to ensure all depressions are breached, keeping in mind that greater distance = greater computing time. Note that the DEM is *not* breached in order of lowest elevation to greatest, nor is it breached sequentially (order is unknown, but the raster is presumably searched in some grid pattern for depressions). This means that flow paths may need to cross multiple depressions, especially in low relief areas.

`streams`	Optionally, the path to the polylines shapefile or geopackage file containing streams, which can be used to improve hydrological accuracy when using poor quality DEMs but decent accuracy stream networks.
`burn_dist`	The number of units (in DEM units) to use for burning-in the stream network.
`save_path`	An optional path in which to save the processed DEM. If left NULL will save it in the same directory as the provided DEM or, if the DEM is a terra object, return only terra objects.
`n.cores`	The maximum number of cores to use. Leave NULL to use all cores minus 1.
`force_update_wbt`	Whitebox Tools is by default only downloaded if it cannot be found on the computer, and no check are performed to ensure the local version is current. Set to TRUE if you know that there is a new version and you would like to use it.

Details

Relies on two WhiteboxTools functions: `whitebox::wbt_fill_single_cell_pits()` and `whitebox::wbt_breach_depre`
If the parameter `streams` is specified, a depression is burned into the DEM after running fill_single_cell_pits and before breaching depressions.

Value

A hydro-processed DEM returned as a terra object and saved to disk if `save_path` is not null.

Author(s)

Ghislain de Laplante (gdela069@uottawa.ca or ghislain.delaplante@yukon.ca)

Examples

```
# Running with terra objects:
res <- hydroProcess(DEM = elev,
  breach_dist = 500,
  streams = streams,
  n.cores = 2)

terra::plot(res)

# Running with file paths:
res <- hydroProcess(DEM = system.file("extdata/dem.tif", package = "SAiVE"),
  breach_dist = 500,
  streams = system.file("extdata/streams.gpkg", package = "SAiVE"),
  n.cores = 2)

terra::plot(res)
```

modelMatch *Find machine learning models for use in caret*

Description

[Experimental]

As of 2023-06-15, there are 238 different machine learning models which can be used with the CARET package. As evaluating model performance is time consuming, selecting a subset of models to test prior to deciding on which model to use is essential. This function aims to facilitate this process by matching models according to their Jaccard similarity, in a process inspired by this section in the CARET e-book. Model data is fetched from here. The result of this function can then be passed to `spatPredict()` to further refine model selection.

Usage

```
modelMatch(model, type = "match", similarity = 0.7)
```

Arguments

`model`	The abbreviation or short name of the model you'd like to match, taken from here.
`type`	The type of model. You can match the input `model` type with "match", or select from dual-purpose models ("dual"), regression models only ("regression"), or classification models only ("classification").
`similarity`	The similarity threshold to use as a numeric value from 0 to 1. Models with a similarity score greater than this will be returned.

Details

This function requires internet access to get an up-to-date list of models.

Value

A data.frame of models meeting the requested similarity threshold along with the model abbreviations that can be passed to `caret::train()` or to function `spatPredict()`.

Author(s)

Ghislain de Laplante (gdela069@uottawa.ca or ghislain.delaplante@yukon.ca)

Examples

```
# Find models similar to 'ranger'
modelMatch("ranger")

# Find only models with a similarity > 0.8 to 'ranger'
```

```
modelMatch("ranger", similarity = 0.8)
```

`permafrost` *Permafrost data*

Description

A small data set of permafrost type with correlated terrain attributes

Usage

```
permafrost
```

Format

`permafrost`:
A data frame with 400 rows and 8 columns

Source

Permafrost type classification from central Yukon, with corresponding values derived from the Canadian Digital Elevation Model.

`permafrost_polygons` *Polygons of permafrost occurrence and type*

Description

Polygons of permafrost occurrence and type

Usage

```
permafrost_polygons
```

Format

`permafrost_polygons`:
A geopackage file loaded as a terra spatVector.

Source

Permafrost classification polygons created from ground observations and geophysics in central Yukon.

`slope` *Raster of slope angle*

Description

Raster of slope angle

Usage

```
slope
```

Format

`slope`:
A tif file loaded as a terra spatRaster

Source

Derived from the Canadian Digital Elevation Model DEM

`solrad` *Raster of solar radiation*

Description

Raster of solar radiation

Usage

```
solrad
```

Format

`solrad`:
A tif file loaded as a terra spatRaster

Source

Derived from the Canadian Digital Elevation Model DEM using the ArcGIS Area Solar Radiation tool

spatPredict *Predict spatial variables using machine learning*

Description

[Stable]

Function to facilitate the prediction of spatial variables using machine learning, including the selection of a particular model and/or model parameters from several user-defined options. Both classification and regression is supported, though please ensure that the models passed to the parameter `methods` are suitable.

Note that you may need to acquiesce to installing supplementary packages, depending on the model types chosen and whether or not these have been run before; this function may not be 'set and forget'.

It is possible to specify multiple machine learning methods (the `methods` parameter) as well as method-specific parameters (the `trainControl` parameter) if you wish to test multiple options and select the best one. To facilitate method selection, refer to function `modelMatch()`. If you are unsure of the best model to use, you can use the `fastCompare` parameter to quickly compare models and select the best one based on accuracy. If you wish to use a single model and/or trainControl object, you can pass a single string to `methods` and a single trainControl object to `trainControl`.

Warning options are changed for this function only to show all warnings as they occur and reset back to their original state upon function completion (a test is done first to ensure it can be reset). This is to ensure that any warnings when running models are shown in sequence with the messages indicating the progress of the function, especially when running multiple models and/or trainControl options.

Usage

```
spatPredict(
  features,
  outcome,
  poly_sample = 1000,
  trainControl,
  methods,
  fastCompare = TRUE,
  fastFraction = NULL,
  thinFeatures = TRUE,
  predict = FALSE,
  n.cores = NULL,
  save_path = NULL
)
```

Arguments

`features`	Independent variables. Must be either a NAMED list of terra spatRasters or a multi-layer (stacked) spatRaster (c(rast1, rast2). All layers must all have the same cell size, alignment, extent, and crs. These rasters should include the

	training extent (that covered by the spatVector in `outcome`) as well as the desired extrapolation extent.
`outcome`	Dependent variable, as a terra spatVector of points or polygons with a single attribute table column (of class integer, numeric or factor). The class of this column dictates whether the problem is approached as a classification or regression problem; see details. If specifying polygons, stratified random sampling will be done with `poly_sample` number of points per unique polygon value.
`poly_sample`	If passing a polygon SpatVector to `outcome`, the number of points to generate from the polygons for each unique polygon value.
`trainControl`	Parameters used to control training of the machine learning model, created with `caret::trainControl()`. Passed to the `trControl` parameter of `caret::train()`. If specifying multiple methods in `methods` you can use a single `trainControl` which will apply to all `methods`, or pass multiple variations to this argument as a list with names matching the names of `methods` (one element for each model specified in methods).
`methods`	A string specifying one or more classification/regression methods(s) to use. Passed to the `method` parameter of `caret::train()`. If specifying more than one method they will all be passed to `caret::resamples()` to compare method performance. Then, if `predict = TRUE`, the method with the highest overall accuracy will be selected to predict the raster surface across the extent of `features`. A different `trainControl` parameter can be used for each method in `methods`.
`fastCompare`	If specifying multiple methods in `methods` or one method with multiple `trainControl` objects, should the points in `outcome` be sub-sampled for the comparison step? The selected method will be trained on the full `outcome` data set after selection. This only applies if `methods` is length > 3, with behavior further modified by fastFraction.
`fastFraction`	The fraction of points to use for the method comparison step (final training and testing is always done on the full data set) if `fastCompare` is TRUE and multiple methods . Default NULL ranges from 1 for 5000 or fewer points to 0.1 for 50 000 or more points. You can also set this to any value between 0 and 1 to override this behavior.
`thinFeatures`	Should random forest selection using `VSURF::VSURF()` be used in an attempt to remove irrelevant variables?
`predict`	TRUE will apply the trained model to the full extent of `features` and return a raster saved to `save_path`.
`n.cores`	The maximum number of cores to use. Leave NULL to use all cores minus 1.
`save_path`	The path (folder) to which you wish to save the predicted raster. Not used unless `predict = TRUE`.

Details

This function partly operates as a convenient means of passing various parameters to the `caret::train()` function, enabling the user to rapidly trial different model types and parameter sets. In addition, pre-processing of data can optionally be done using `VSURF::VSURF()` (parameter `thinFeatures`) which can decrease the time to run models by removing superfluous parameters.

Value

If passing only one method to the `method` argument: the outcome of the VSURF variable selection process (if `thinFeatures` is TRUE), the training and testing data.frames, the fitted model, model performance statistics, and the final predicted raster (if `predict` = TRUE).

If passing multiple methods to the `method` argument: the outcome of the VSURF variable selection process (if `thinFeatures` is TRUE), the training and testing data.frames, character vectors for failed methods, methods which generated a warning, and what those errors and warnings were, model performance comparison (if methods includes more than one method), the selected method, the trained model performance statistics, and the final predicted raster (if `predict` = TRUE).

In either case, the predicted raster is written to disk if `save_path` is specified.

Model testing, comparison, and reported metrics

After extracting raster values at *n* points from the `features` rasters the point values are split spatially into training and testing sets along a 70/30 split. This is accomplished by creating a grid (1000*1000) of polygons over the extent of the points and randomly assigning polygons to training or testing sets. Points within these polygons are then assigned to the corresponding set, ensuring that the training and testing sets are spatially independent.

Method for selecting the best model:

When specifying multiple model types in`methods`, each model type and `trainControl` pair (if `trainControl` is a list of length equal to `methods`) is run using `caret::train()`. To speed things up you can use `fastCompare` = TRUE. Models are then compared on their 'accuracy' metric as output by `caret::resamples()` when run on the testing partition, and the highest-performing model is selected. If `fastCompare` is TRUE, this model is then run on the complete data set provided in `outcome`. Model statistics are returned upon function completion, which allows the user to select their own 'best performing' model based on other criteriaif desired.

Balancing classes in outcome (dependent) variable

Models can be biased if they are given significantly more points in one outcome class vs others, and best practice is to even out the number of points in each class. If extracting point values from a vector or raster object and passing a points vector object to this function, a simple way to do that is by using the "strata" parameter if using `terra::spatSample()`. If working directly from points, `caret::downSample()` and `caret::upSample()` can be used. See this link for more information. Note that if passing a polygons object to this function stratified random sampling will automatically be performed.

Classification or regression

Whether this function treats your inputs as a classification or regression problem depends on the class attached to the outcome variable. A class `factor` will be treated as a classification problem while all other classes will be treated as regression problems.

Author(s)

Ghislain de Laplante (gdela069@uottawa.ca or ghislain.delaplante@yukon.ca)

Examples

```
# These examples can take a while to run!

# Install packages underpinning examples
rlang::check_installed("ranger", reason = "required to run example.")
rlang::check_installed("Rborist", reason = "required to run example.")

# Single model, single trainControl

trainControl <- caret::trainControl(
                method = "repeatedcv",
                number = 2, # 2-fold Cross-validation
                repeats = 2, # repeated 2 times
                verboseIter = FALSE,
                returnResamp = "final",
                savePredictions = "all",
                allowParallel = TRUE)

 outcome <- permafrost_polygons
 outcome$Type <- as.factor(outcome$Type)

result <- spatPredict(features = c(aspect, solrad, slope),
  outcome = outcome,
  poly_sample = 100,
  trainControl = trainControl,
  methods = "ranger",
  n.cores = 2,
  predict = TRUE)

terra::plot(result$prediction)


# Multiple models, multiple trainControl

trainControl <- list("ranger" = caret::trainControl(
                                  method = "repeatedcv",
                                  number = 2,
                                  repeats = 2,
                                  verboseIter = FALSE,
                                  returnResamp = "final",
                                  savePredictions = "all",
                                  allowParallel = TRUE),
                     "Rborist" = caret::trainControl(
                                   method = "boot",
                                   number = 2,
                                   repeats = 2,
                                   verboseIter = FALSE,
                                   returnResamp = "final",
                                   savePredictions = "all",
                                   allowParallel = TRUE)
                                   )
```

```
result <- spatPredict(features = c(aspect, solrad, slope),
  outcome = outcome,
  poly_sample = 100,
  trainControl = trainControl,
  methods = c("ranger", "Rborist"),
  n.cores = 2,
  predict = TRUE)

terra::plot(result$prediction)
```

streams *Lines representing streams*

Description

Lines representing streams

Usage

```
streams
```

Format

`streams`**:**
A geopackage file loaded as a terra spatVector

Source

A subset of the Yukon CANVEC water flow lines found here

thinFeatures *Remove irrelevant predictor variables*

Description

Uses `VSURF::VSURF()` to build random forests and remove irrelevant predictor variables from a data.frame containing an outcome variable and 2 or more predictor variables.

Usage

```
thinFeatures(data, outcome_col, n.cores = NULL)
```

Arguments

`data`	A data.frame containing a column for the outcome variable and *n* columns for predictor variables.
`outcome_col`	The name of the outcome variable column.
`n.cores`	The maximum number of cores to use. Leave NULL to use all cores minus 1.

Value

A list of two data.frames: the outcome of the VSURF algorithm and the data after applying the VSURF results (rows removed if applicable)

Examples

```
# thinFeatures on 'permafrost' data set

data(permafrost)
res <- thinFeatures(permafrost, "Type", n.cores = 2)

# Results will vary due to inherent randomness of random forests!
```

`veg` *Raster of vegetation types*

Description

Raster of vegetation types

Usage

```
veg
```

Format

`veg`:
A tif file loaded as a terra spatRaster

Source

A small subset of the North American Land Cover dataset produced by the Commission for Environmental Cooperation, resampled to match cell size of other rasters in this package.

`wbtCheck` *Check WhiteboxTools binaries installation*

Description

Checks for the existence of WhiteboxTools in its default directory and installs it if necessary or if `force = TRUE`.

Usage

```
wbtCheck(force = FALSE)
```

Arguments

`force`	Set TRUE to force update of WhiteboxTools binaries.

Value

Returns the version number of the installed binaries and (if necessary) installs WhiteboxTools in its default location.

Author(s)

Ghislain de Laplante (gdela069@uottawa.ca or ghislain.delaplante@yukon.ca)

Examples

```
#Check if WhiteboxTools binaries are installed. If not, install latest version.
wbtCheck()

# Update WhiteboxTools binaries if they are already installed.
wbtCheck(force = TRUE)
```

Index