Package 'icardaFIGSr'

August 5, 2021

Type Package

Title Subsetting using Focused Identification of the Germplasm Strategy (FIGS)

Version 1.0.1

Description Running Focused Identification of the Germplasm Strategy (FIGS) to make best subsets from Genebank Collection.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports caret, doParallel, dplyr, ggplot2, RGtk2, gWidgets2RGtk2, gWidgets2, foreach, httr, magrittr, methods, plotROC, plyr, raster, reshape2, sp, leaflet

RoxygenNote 7.1.1

Suggests

NeedsCompilation no

Depends R (>= 3.5.0)

Author Khadija Aziz [aut], Zakaria Kehel [aut, cre], Bancy Ngatia [aut], Khadija Aouzal [aut], Zainab Azough [ctb], Amal Ibnelhobyb [ctb], Fawzy Nawar [ctb]

Maintainer Zakaria Kehel <z.kehel@cgiar.org>

Repository CRAN

Date/Publication 2021-08-05 08:40:02 UTC

R topics documented:

`.authenticate` *GUI for getting username and password from user*

Description

GUI for getting username and password from user

Usage

```
.authenticate()
```

Value

No return value

durumDaily *durumDaily*

Description

200 sites from durum wheat collection and their daily climatic data.

Usage

```
data(durumDaily)
```

Format

The data includes the site unique identifier and daily data for 4 climatic variables (tmin, tmax, precipitation and relative humidity)

Examples

```
if(interactive()){
 # Load durum wheat data with their daily climatic variables obtained from ICARDA database
 data(durumDaily)
}
```

durumWC *durumWC*

Description

200 sites from durum wheat collection and their world clim data.

Usage

```
data(durumWC)
```

Format

The data includes the site unique identifier, longitude, latitude and 55 worldclim data worldclim

Examples

```
if(interactive()){
 # Load durum wheat data with world climatic variables obtained from WorldClim database
 data(durumWC)
 }
```

`extractWCdata` *Extracting World Climatic Data*

Description

extractWCdata returns a data frame based on specified climatic variables.

Usage

```
extractWCdata(sites, long, lat, var, res = 2.5)
```

Arguments

`sites`	object of class "data.frame" with coordinates of sites from which to extract data.
`long`	character. Name of column from `sites` with longitude.
`lat`	character. Name of column from `sites` with latitude.
`var`	character. Climatic variable(s) to be extracted: 'tavg', 'tmin', 'tmax', 'prec', 'bio', 'srad', 'vapr', 'wind'
`res`	numeric. Spatial resolution. Default 2.5

Details

A grid can be created with any particular coordinates and used as input for `sites` (see section 'Examples'). `extractWCdata` will use the given coordinates to extract data from the WorldClim2.1 database. The extracted data will most likely contain NA's for sites where climatic data is not available. These should be removed or imputed before using the data to make predictions.

Value

An object of class "data.frame" with specified climatic variables for coordinates in `sites`.

Author(s)

Zakaria Kehel, Fawzy Nawar, Bancy Ngatia, Khadija Aouzal

Examples

```
if(interactive()){
 # Create grid
 sp1 <- seq(-16, 115, length = 10)
 sp2 <- seq(25, 59, length = 10)
 sp <- expand.grid(x = sp1, y = sp2)

 # Extract data using grid
 sp.df0 <- extractWCdata(sp, long = 'x', lat = 'y', var = 'tavg')
 sp.df <- na.omit(sp.df0)
 }
```

FIGS *FIGS subset for wheat sodicity resistance*

Description

FIGS subset for wheat sodicity resistance constructed using the harmonized world soil database HWSD

Usage

```
data(FIGS)
```

Format

A data frame with 201 rows and 15 variables

References

HWSD

Examples

```
if(interactive()){
 data(FIGS)
}
```

getAccessions *Getting Accession Data from ICARDA's Genebank Documentation System*

Description

Return a data frame with accession data for the specified crop.

Usage

```
getAccessions(
  crop = "",
  ori = NULL,
  IG = "",
  doi = FALSE,
  taxon = FALSE,
  collectionYear = FALSE,
  coor = FALSE,
  available = FALSE
)
```

Arguments

crop	character. Crop for which to get accession data. See section 'Details' for available crops or use getCrops function. Default: "".
ori	string. Country of origin using the ISO 3166-1 alpha-3 country codes. Default: NULL.
IG	integer. Unique identifier of accession. Default: "".
doi	boolean. If TRUE , the function will return the digital object identifiers DOI for the accessions. Default: FALSE.
taxon	boolean. If TRUE, the function will return the taxon information of the accessions. Default: FALSE.
collectionYear	boolean. If TRUE, the function will return the year of the collecting mission. Default: FALSE.
coor	boolean. If TRUE, returns only georeferenced accessions containing longitude and latitude. Default: FALSE.
available	boolean. If TRUE, returns the availability of accessions for distribution, Default: FALSE.

Details

Types of crops available include:

- 'Aegilops'
- 'Barley'
- 'Bread wheat'
- 'Chickpea'
- 'Durum wheat'
- 'Faba bean'
- 'Faba bean BPL'
- 'Forage and range'
- 'Lathyrus'
- 'Lentil'
- 'Medicago annual'
- 'Not mandate cereals'
- 'Pisum'
- 'Primitive wheat'
- 'Trifolium'
- 'Vicia'
- 'Wheat hybrids'
- 'Wheat wild relatives'
- 'Wild Cicer'

- 'Wild Hordeum'
- 'Wild Lens'
- 'Wild Triticum'

Alternatively, the list of available crops can be fetched from ICARDA's online server using `getCrops`.

Value

A data frame with accession passport data for specified crop in `crop` from the locations in `ori`.

Author(s)

Khadija Aouzal, Amal Ibnelhobyb, Zakaria Kehel, Fawzy Nawar

Examples

```
if(interactive()){
 # Obtain accession data for durum wheat
 durum <- getAccessions(crop = 'Durum wheat', coor = TRUE)
 }
```

---

getCrops *Crops Available in ICARDA's Genebank*

Description

this function allows to obtain a list of crops available in ICARDA's Genebank Documentation System, it returns a list with codes and names of available crops.

Usage

```
getCrops()
```

Details

The crop codes and names are fetched from ICARDA's online server.

Value

A list containing all crops available in ICARDA's Genebank Documentation System.

Author(s)

Zakaria Kehel, Fawzy Nawar

Examples

```
if(interactive()){
 # Get list of available crops
 crops <- getCrops()
 }
```

getDaily *Extracting Daily Climatic Variables*

Description

this function extracts daily values of climatic variables from ICARDA Data, it returns a list or data frame based on specified climatic variables. Each variable will have 365 values for each day of the calendar year.

Usage

```
getDaily(sites, var, cv = FALSE)
```

Arguments

sites	character. Names of sites from which to extract data.
var	character. Climatic variable(s) to be extracted.
cv	boolean. If TRUE, returns a data frame with coefficient of variation for each variable for each day of the calendar year. Default: FALSE.

Details

ICARDA data has to be accessible either from a local directory on the computer or from an online repository. getDaily will extract the climatic variables specified in var for the sites specified in sites.

For daily data, the function extracts average daily values starting from the first day of the calendar year, i.e. January 1, until the last day of the calendar year, i.e. December 31. Thus, 365 columns with daily values are created for each variable.

Value

An object with specified climatic variables for names in sites.

If cv = TRUE, the object is a list containing two data frames: the first one with average daily values of climatic variables, and the second one with daily coefficient of variation for each climatic variable.

If cv = FALSE, the object is a data frame with average daily values of climatic variables.

Author(s)

Zakaria Kehel, Bancy Ngatia

See Also

cast

Examples

```
if(interactive()){
 # Extract daily data for durum wheat
 durum <- getAccessions(crop = 'Durum wheat', coor = TRUE)
 daily <- getDaily(sites = levels(as.factor(durum$SiteCode)),
                   var = c('tavg', 'prec', 'rh'), cv = TRUE)

 # Get data frame with coefficient of variation from list object
 # returned (when cv = TRUE)
 daily.cv <- daily[[2]]
 }
```

`getGrowthPeriod` *Calculating Growing Degree Days and Lengths of Growth Stages for Various Crops Using Onset Data from ICARDA's Database*

Description

Calculates growing degree days (GDD) as well as cumulative GDD, and returns a list of various data frames based on specified arguments.

Usage

```
getGrowthPeriod(sitecode, crop, base, max, gdd = FALSE)
```

Arguments

`sitecode`	expression. Vector with names of sites from which to extract onset data.
`crop`	character. Type of crop in ICARDA database. See section 'Details' for crops which have calculations available.
`base`	integer. Minimum temperature constraint for the crop.
`max`	integer. Maximum temperature constraint for the crop.
`gdd`	boolean. If `TRUE`, returns a data frame containing calculated GDD and accumulated GDD together with climatic variables used for the calculations. Default: FALSE.

Details

Growing degree days for various crops are calculated using average daily minimum and maximum temperature values obtained from onset data. The temperature constraints specified in `base` and `max` are first applied before the calculations are done. These constraints ensure very low or high temperatures which prevent growth of a particular crop are not included. Crops for which GDD calculations are available include: 'Durum wheat', 'Bread wheat', 'Barley', 'Chickpea', 'Lentil'. Each of these can be supplied as options for the argument `crop`. Cumulative GDD values determine the length of different growing stages. Growing stages vary depending on the type of crop. Durum wheat, bread wheat and barley have five growth stages, i.e. beginning of heading, beginning and

completion of flowering, and beginning and completion of grain filling. Chickpea and lentil have four growth stages, i.e. beginning of flowering, completion of 50 The length of the full growth cycle of the crop for each site is also given in the output data frame.

Value

A list object with different data frames depending on specified option in gdd. If gdd = TRUE, the object is a list containing three data frames: the first one with lengths of different growing stages, the second one with original onset data with phenological variables, and the third one with calculated GDD and accumulated GDD for the sites specified in sitecode. If gdd = FALSE, the object is a list containing two data frames: the first one with lengths of different growing stages, and the second one with original onset data with phenological variables for the sites specified in sitecode.

Author(s)

Khadija Aouzal, Zakaria Kehel, Bancy Ngatia

Examples

```
if(interactive()){
 # Calculate GDD for durum wheat
 data(durumDaily)
 growth <- getGrowthPeriod(sitecode = durumDaily$site_code,
                           crop = 'Durum wheat', base = 0,
                           max = 35, gdd = TRUE)

 # Get data frame with lengths of growth stages from list
 # object returned
 growth.lengths <- growth[[1]]

 # Get data frame with phenotypic variables from list
 # object returned
 growth.pheno <- growth[[2]]

 # Get data frame with GDD, cumulative GDD and climatic
 # variables from list object returned (when gdd = TRUE)
 growth.gdd <- growth[[3]]
 }
```

---

getMetrics *Performance Measures*

Description

this function allows to obtain performance measures from Confusion Matrix, it returns a data frame containing performance measures from the confusion matrix given by the caret package.

Usage

```
getMetrics(y, yhat, classtype)
```

Arguments

y	expression. The class variable.
yhat	expression. The vector of predicted values.
classtype	character or numeric. The number of levels in y.

Details

getMetrics works with target variables that have two, three, four, six or eight classes.

The function relies on the caret package to obtain the confusion matrix from which performance measures are extracted. It can be run for several algorithms, and the results combined into one data frame for easier comparison (see section 'Examples').

Predictions have to be obtained beforehand and used as input for yhat. The predict.train function in caret should be run without argument type when obtaining the predictions.

Value

Outputs an object with performance measures calculated from the confusion matrix given by the caret package. A data frame is the resulting output with the first column giving the name of the performance measure, and the second column giving the corresponding value.

Author(s)

Zakaria Kehel, Bancy Ngatia, Khadija Aziz

See Also

confusionMatrix

Examples

```
if(interactive()){
# Obtain predictions from previous models

data(septoriaDurumWC)
split.data <- splitData(septoriaDurumWC, seed = 1234, y = "ST_S", p = 0.7)
data.train <- split.data$trainset
data.test <- split.data$testset

knn.mod <- tuneTrain(data = septoriaDurumWC,y = 'ST_S',method = 'knn',positive = 'R')
nnet.mod <- tuneTrain(data = septoriaDurumWC,y = 'ST_S',method = 'nnet',positive = 'R')

pred.knn <- predict(knn.mod$Model, newdata = data.test[ , -1])
pred.nnet <- predict(nnet.mod$Model, newdata = data.test[ , -1])

metrics.knn <- getMetrics(y = data.test$ST_S, yhat = pred.knn, classtype = 2)
metrics.nnet <- getMetrics(y = data.test$ST_S, yhat = pred.nnet, classtype = 2)
}
```

getMetricsPCA *Performance Measures with PCA pre-processing*

Description

getMetricsPCA allows to obtain performance measures from Confusion Matrix for algorithms with PCA pre-processing,it returns a data frame containing performance measures from the confusion matrix given by the `caret` package when algorithms have been run with PCA pre-processing.

Usage

```
getMetricsPCA(yhat, y, classtype, model)
```

Arguments

`yhat`	expression. The vector of predicted values.
`y`	expression. The class variable.
`classtype`	character or numeric. The number of levels in `y`.
`model`	expression. The model object to which output of the model has been assigned.

Details

Works with target variables that have two, three, four, six or eight classes. Similar to `getMetrics` but used in the case where models have been run with PCA specified as an option for the `preProcess` argument in the `train` function of `caret`.

Value

Outputs an object with performance measures calculated from the confusion matrix given by the `caret` package. A data frame is the resulting output with the first column giving the name of the performance measure, and the second column giving the corresponding value.

Author(s)

Khadija Aziz, Zainab Azough, Zakaria Kehel, Bancy Ngatia

See Also

`confusionMatrix`, `predict.train`

Examples

```
if(interactive()){
 # Obtain predictions from several previously run models
 dataX <- subset(data, select = -y)
 pred.knn <- predict(model.knn, newdata = dataX)
 pred.rf <- predict(model.rf, newdata = dataX)
```

```
# Get metrics for several algorithms
metrics.knn <- getMetricsPCA(y = data$y, yhat = pred.knn,
                             classtype = 2, model = model.knn)
metrics.rf <- getMetricsPCA(y = data$y, yhat = pred.rf,
                            classtype = 2, model = model.rf)

# Indexing for 2-class models to remove extra column with
# names of performance measures
metrics.all <- cbind(metrics.knn, metrics.rf[ , 2])

# No indexing needed for 3-, 4-, 6- or 8-class models
metrics.all <- cbind(metrics.knn, metrics.rf)
}
```

`getOnset` *Extracting Daily Climatic Variables Based on Onset of Planting*

Description

this function Extracts Daily values of climatic variables from remote ICARDA data based on Onset of Planting, it returns a list based on specified climatic variables. Each variable will have 365 values for each day of the (onset) year beginning with planting day.

Usage

```
getOnset(sites, crop, var, cv = FALSE)
```

Arguments

`sites`	character. Names of sites from which to extract data.
`crop`	character. Crop code in ICARDA database. See section 'Details' for a list of crops.
`var`	character. Climatic variable(s) to be extracted.
`cv`	boolean. If `TRUE`, returns a data frame with coefficient of variation for each variable for each day of the onset year. Default: FALSE.

Details

Similar to `getDaily` except the extracted data is based on 365 days starting from the onset of planting. Crops available in ICARDA's genebank documentation system include the following:

- 'ICAG' = Aegilops
- 'ICB' = Barley
- 'ICBW' = Bread wheat
- 'ILC' = Chickpea
- 'ICDW' = Durum wheat

- 'ILB' = Faba bean
- 'BPL' = Faba bean BPL
- 'IFMI' = Forage and range
- 'IFLA' = Lathyrus
- 'ILL' = Lentil
- 'IFMA' = Medicago annual
- 'IC' = Not mandate cereals
- 'IFPI' = Pisum
- 'ICPW' = Primitive wheat
- 'IFTR' = Trifolium
- 'IFVI' = Vicia
- 'ICWH' = Wheat hybrids
- 'ICWW' = Wheat wild relatives
- 'ILWC' = Wild Cicer
- 'ICWB' = Wild Hordeum
- 'ILWL' = Wild Lens
- 'ICWT' = Wild Triticum

Alternatively, the list of available crops can be fetched from ICARDA's online server using `getCrops`.

Value

An object of class "data.frame" with specified climatic variables for names in `sites`.

If `cv = TRUE`, the object is a list containing three data frames: the first one with average daily values of climatic variables, the second one with daily coefficient of variation for each climatic variable, and the third one with phenotypic variables and number of day in calendar year when each occurs at the sites specified in `sites`.

If `cv = FALSE`, the object is a list containing two data frames: the first one with average daily values of climatic variables, and the second one with phenotypic variables and number of day in calendar year when each occurs at the sites specified in `sites`.

Author(s)

Khadija Aouzal, Amal Ibnelhobyb, Zakaria Kehel, Bancy Ngatia

See Also

`dcast`, `getCrops`

Examples

```
if(interactive()){
 # Extract onset data for durum wheat
 durum <- getAccessions(crop = 'Durum wheat', coor = TRUE)
 onset <- getOnset(sites = levels(as.factor(durum$SiteCode)), crop = 'ICDW',
                   var = c('tavg', 'prec', 'rh'), cv = TRUE)

 # Get data frame with climatic variables from list object returned
 onset.clim <- onset[[1]]

 # Get data frame with coefficient of variation from list object
 # returned (when cv = TRUE)
 onset.cv <- onset[[2]]

 # Get data frame with phenotypic variables from list object returned
 onset.pheno <- onset[[3]]
 }
```

`getTraits` *Getting Traits Associated with Crops from the ICARDA's Genebank Documentation System*

Description

Return a data frame containing traits associated with a particular crop, their description and related identifiers.

Usage

```
getTraits(crop)
```

Arguments

`crop`	character. Crop for which to get available traits.

Details

`getTraits` returns a data frame of traits together with their IDs and coding system used for each trait.

Possible inputs for `crop` include:

- 'Aegilops'
- 'Barley'
- 'Bread wheat'
- 'Chickpea'
- 'Durum wheat'
- 'Faba bean'

- 'Faba bean BPL'
- 'Forage and range'
- 'Lathyrus'
- 'Lentil'
- 'Medicago annual'
- 'Not mandate cereals'
- 'Pisum'
- 'Primitive wheat'
- 'Trifolium'
- 'Vicia'
- 'Wheat hybrids'
- 'Wheat wild relatives'
- 'Wild Cicer'
- 'Wild Hordeum'
- 'Wild Lens'
- 'Wild Triticum'

A list of available crops to use as input for `crop` can also be obtained from ICARDA's online server using `getCrops`.

Value

A data frame with traits that are associated with the crop specified in `crop`.

Author(s)

Khadija Aouzal, Amal Ibnelhobyb, Zakaria Kehel, Fawzy Nawar

Examples

```
if(interactive()){
 # Get traits for bread wheat
 breadTraits <- getTraits(crop = 'Bread wheat')
}
```

getTraitsData *Getting Trait Values of Accessions for a Specific Trait*

Description

Return a data frame with observed values of accessions for associated Trait

Usage

```
getTraitsData(IG, traitID)
```

Arguments

IG	factor. Unique identifier of accession.
traitID	integer. Unique identifier of trait (from getTraits).

Details

Possible inputs for traitID can be found using the getTraits function (see section 'Examples').

Value

A data frame with scores for the trait specified in traitID for the accessions given in IG.

Author(s)

Khadija Aouzal, Amal Ibnelhobyb, Zakaria Kehel, Fawzy Nawar

Examples

```
if(interactive()){
 # Check trait ID for septoria and get septoria data for durum wheat
 durum <- getAccessions(crop = 'Durum wheat', coor = TRUE)
 durumTraits <- getTraits(crop = 'Durum wheat')
 septoria <- getTraitsData(IG = durum$IG, traitID = 145)
}
```

mapAccessions *Plotting Accessions on Map.*

Description

this function returns a map with points showing where accessions are located.

Usage

```
mapAccessions(df, long, lat, y = NULL)
```

Arguments

df	object of class "data.frame" with coordinates of accessions and target variable.
long	character. Column name from df representing longitude.
lat	character. Column name from df representing latitude.
y	Default: NULL, column name from df representing the target variable.

Value

A world map with plotted points showing locations of accessions.

Author(s)

Khadija Aouzal, Zakaria Kehel

Examples

```
if(interactive()){
 # Loading FIGS subset for wheat sodicity resistance
 data(FIGS)
 # World Map showing locations of accessions
 mapAccessions(df = FIGS, long = "Longitude", lat = "Latitude")

 # Map plotting locations of accessions with points coloured
 # based on a gradient scale of SodicityIndex values
 mapAccessions(FIGS, long = "Longitude", lat = "Latitude",
               y = "SodicityIndex")
 # Map plotting locations of accessions with points
 # coloured based on levels of y
 mapAccessions(FIGS, long = "Longitude", lat = "Latitude",
 y = "PopulationType")
 }
```

`modelingSummary` *Get modeling metrics*

Description

modelingSummary is an automatic function for modeling data, it returns a dataframe containing the metrics of the modeling using five machine learning algorithms: KNN, SVM, RF, NNET, and Bcart. This function is based on spliData, tuneTrain, predict, and getMetrics functions.

Usage

```
modelingSummary(
  data,
  y,
  p = 0.7,
  length = 10,
  control = "repeatedcv",
  number = 10,
  repeats = 10,
  process = c("center", "scale"),
  summary = multiClassSummary,
  positive,
  parallelComputing = FALSE,
  classtype,
  ...
)
```

Arguments

`data`	object of class "data.frame" with target variable and predictor variables.
`y`	character. Target variable.
`p`	numeric. Proportion of data to be used for training. Default: 0.7
`length`	integer. Number of values to output for each tuning parameter. If `search = "random"` is passed to `trainControl` through `...`, this becomes the maximum number of tuning parameter combinations that are generated by the random search. Default: 10.
`control`	character. Resampling method to use. Choices include: "boot", "boot632", "optimism_boot", "boot_all", "cv", "repeatedcv", "LOOCV", "LGOCV", "none", "oob", timeslice, "adaptive_cv", "adaptive_boot", or "adaptive_LGOCV". Default: "repeatedcv". See `train` for specific details on the resampling methods.
`number`	integer. Number of cross-validation folds or number of resampling iterations. Default: 10.
`repeats`	integer. Number of folds for repeated k-fold cross-validation if "repeatedcv" is chosen as the resampling method in `control`. Default: 10.

process	character. Defines the pre-processing transformation of predictor variables to be done. Options are: "BoxCox", "YeoJohnson", "expoTrans", "center", "scale", "range", "knnImpute", "bagImpute", "medianImpute", "pca", "ica", or "spatialSign". See preProcess for specific details on each pre-processing transformation. Default: c('center', 'scale').
summary	expression. Computes performance metrics across resamples. For numeric y, the mean squared error and R-squared are calculated. For factor y, the overall accuracy and Kappa are calculated. See trainControl and defaultSummary for details on specification and summary options. Default: multiClassSummary.
positive	character. The positive class for the target variable if y is factor. Usually, it is the first level of the factor.
parallelComputing	logical. indicates whether to also use the parallel processing. Default: False
classtype	integer.indicates the number of classes of the traits.
...	additional arguments to be passed to createDataPartition, trainControl and train functions in the package caret.

Details

Types of classification and regression models available for use with tuneTrain can be found using names(getModelInfo()). The results given depend on the type of model used.

Value

A dataframe contains the metrics of the modeling of five machine learning algorithms: KNN, SVM, RF, NNET, and Bcart.

tuneTrain relies on package caret to perform the modeling.

Author(s)

Zakaria Kehel, Khadija Aziz

See Also

createDataPartition, trainControl, train, predict.train, confusionMatrix

Examples

```
if(interactive()){
 data(septoriaDurumWC)
 models <- modelingSummary(data = septoriaDurumWC, y = "ST_S", positive = "R", classtype = 2)
}
```

septoriaDurumWC *septoriaDurumWC*

Description

A sample data including daily data for 4 climatic variables (tmin, tmax, precipitation and relative humidity) and evaluation for Septoria Tritici

Usage

```
data(septoriaDurumWC)
```

Format

200 sites from durum wheat collection and their daily climatic data and evaluation for Septoria Tritici.

Examples

```
if(interactive()){
 #Load durum wheat data with septoria scores and climatic variables obtained from WorldClim database
 data(septoriaDurumWC)
}
```

splitData *Splitting Data*

Description

this function splits the Data into Train and Test Sets, it returns a list containing two data frames for the train and test sets.

Usage

```
splitData(data, seed = NULL, y, p, ...)
```

Arguments

data	object of class "data.frame" with target variable and predictor variables.
seed	integer. Values for the random number generator. Default: NULL.
y	character. Target variable.
p	numeric. Proportion of data to be used for training.
...	additional arguments to be passed to createDataPartition function in caret package to control the way the data is split.

Details

splitData relies on the createDataPartition function from the caret package to perform the data split.

If y is a factor, the sampling of observations for each set is done within the levels of y such that the class distributions are more or less balanced for each set.

If y is numeric, the data is split into groups based on percentiles and the sampling done within these subgroups. See createDataPartition for more details on additional arguments that can be passed.

Value

A list with two data frames: the first as train set, and the second as test set.

Author(s)

Zakaria Kehel, Bancy Ngatia

See Also

createDataPartition

Examples

```
if(interactive()){
 # Split the data into 70/30 train and test sets for factor y
 data(septoriaDurumWC)
 split.data <- splitData(septoriaDurumWC, seed = 1234,
                         y = 'ST_S', p = 0.7)

 # Get training and test sets from list object returned
 trainset <- split.data$trainset
 testset <- split.data$testset
}
```

---

tuneTrain *Tuning and Training the Data*

Description

tuneTrain splits the Data, it is an automatic function for tuning, training, and making predictions, it returns a list containing a model object, data frame and plot.

Usage

```
tuneTrain(
  data,
  y,
  p = 0.7,
  method = method,
  parallelComputing = FALSE,
  length = 10,
  control = "repeatedcv",
  number = 10,
  repeats = 10,
  process = c("center", "scale"),
  summary = multiClassSummary,
  positive,
  ...
)
```

Arguments

data	object of class "data.frame" with target variable and predictor variables.
y	character. Target variable.
p	numeric. Proportion of data to be used for training. Default: 0.7
method	character. Type of model to use for classification or regression.
parallelComputing	logical. indicates whether to also use the parallel processing. Default: False
length	integer. Number of values to output for each tuning parameter. If search = "random" is passed to trainControl through ..., this becomes the maximum number of tuning parameter combinations that are generated by the random search. Default: 10.
control	character. Resampling method to use. Choices include: "boot", "boot632", "optimism_boot", "boot_all", "cv", "repeatedcv", "LOOCV", "LGOCV", "none", "oob", timeslice, "adaptive_cv", "adaptive_boot", or "adaptive_LGOCV". Default: "repeatedcv". See train for specific details on the resampling methods.
number	integer. Number of cross-validation folds or number of resampling iterations. Default: 10.
repeats	integer. Number of folds for repeated k-fold cross-validation if "repeatedcv" is chosen as the resampling method in control. Default: 10.
process	character. Defines the pre-processing transformation of predictor variables to be done. Options are: "BoxCox", "YeoJohnson", "expoTrans", "center", "scale", "range", "knnImpute", "bagImpute", "medianImpute", "pca", "ica", or "spatialSign". See preProcess for specific details on each pre-processing transformation. Default: c('center', 'scale').
summary	expression. Computes performance metrics across resamples. For numeric y, the mean squared error and R-squared are calculated. For factor y, the overall accuracy and Kappa are calculated. See trainControl and defaultSummary for details on specification and summary options. Default: multiClassSummary.

positive	character. The positive class for the target variable if y is factor. Usually, it is the first level of the factor.
...	additional arguments to be passed to createDataPartition, trainControl and train functions in the package caret.

Details

Types of classification and regression models available for use with tuneTrain can be found using names(getModelInfo()). The results given depend on the type of model used.

For classification models, class probabilities and ROC curve are given in the results. For regression models, predictions and residuals versus predicted plot are given. y should be converted to either factor if performing classification or numeric if performing regression before specifying it in tuneTrain.

Value

A list object with results from tuning and training the model selected in method, together with predictions and class probabilities. The training and test data sets obtained from splitting the data are also returned.

If y is factor, class probabilities are calculated for each class. If y is numeric, predicted values are calculated.

A ROC curve is created if y is factor. Otherwise, a plot of residuals versus predicted values is created if y is numeric.

tuneTrain relies on packages caret, ggplot2 and plotROC to perform the modelling and plotting.

Author(s)

Zakaria Kehel, Bancy Ngatia, Khadija Aziz

See Also

createDataPartition, trainControl, train, predict.train, ggplot, geom_roc, calc_auc

Examples

```
if(interactive()){
 data(septoriaDurumWC)
 knn.mod <- tuneTrain(data = septoriaDurumWC,y = 'ST_S',method = 'knn',positive = 'R')

 nnet.mod <- tuneTrain(data = septoriaDurumWC,y = 'ST_S',method = 'nnet',positive = 'R')

}
```

varimpPred *Variable Importance and Predictions*

Description

varimpPred calculates Variable Importance and makes predictions, it returns a list containing a data frame of variable importance scores, predictions or class probabilities, and corresponding plots.

Usage

```
varimpPred(
  newdata,
  y,
  positive,
  model,
  scale = FALSE,
  auc = FALSE,
  predict = FALSE,
  ...
)
```

Arguments

newdata	object of class "data.frame" having test data.
y	character. Target variable.
positive	character. The positive class for the target variable if y is factor. Usually, it is the first level of the factor.
model	expression. The model object returned after training a model on training data.
scale	boolean. If TRUE, scales the variable importance values to between 0-100. Default: FALSE.
auc	boolean. If TRUE, calculates the area under the ROC curve and returns the value. Default: FALSE.
predict	boolean. If TRUE, calculates class probabilities and returns them as a data frame. Default: FALSE
...	additional arguments to be passed to varImp function in the package caret.

Details

The importance measure for each variable is calculated based on the type of model.

For example for linear models, the absolute value of the t-statistic of each parameter is used in the importance calculation.

For classification models, with the exception of classification trees, bagged trees and boosted trees, a variable importance score is calculated for each class. See varImp for details on model-specific metrics.

varimpPred can be used to obtain either variable importance metrics, predictions, class probabilities, or a combination of these.

For classification models with predict = TRUE, class probabilities and ROC curve are given in the results.

For regression models with predict = TRUE, predictions and residuals versus predicted plot are given.

Value

A list object with importance measures for variables in newdata, predictions for regression models, class probabilities for classification models, and corresponding plots.

newdata should be either the test data that remains after splitting whole data into training and test sets, or a new data set different from the one used to train the model.

If y is factor, class probabilities are calculated for each class. If y is numeric, predicted values are calculated.

A ROC curve is created if predict = TRUE and y is factor. Otherwise, a plot of residuals versus predicted values is created if y is numeric.

varimpPred relies on packages caret, ggplot2 and plotROC to perform the calculations and plotting.

Author(s)

Zakaria Kehel, Bancy Ngatia, Khadija Aziz, Zainab Azough

See Also

varImp, predict.train, ggplot, geom_roc, calc_auc

Examples

```
if(interactive()){
 # Calculate variable importance for classification model
 data("septoriaDurumWC")
 knn.mod <- tuneTrain(data = septoriaDurumWC,y = 'ST_S',method = 'knn')
 testdata <- knn.mod$`Test Data`
knn.varimp<- varimpPred(newdata = testdata, y='ST_S', positive = 'R', model = knn.mod$Model)
 knn.varimp

 # Calculate variable importance and obtain class probabilities
 data("septoriaDurumWC")
 svm.mod <- tuneTrain(data = septoriaDurumWC, y = 'ST_S',method = 'svmLinear2',
                  predict = TRUE, positive = 'R',summary = twoClassSummary)
 testdata <- svm.mod$`Test Data`
 svm.varimp <- varimpPred(newdata = testdata, y = 'ST_S',
                          positive = 'R', model = svm.mod$Model,
                          ROC = TRUE, predict = TRUE)
 svm.varimp
 # Obtain variable importance plot for only first 20 variables
 # with highest measure
```

```
svm.varimp <- varimpPred(newdata = testdata, y = 'ST_S',
                         positive = 'R', model = svm.mod$Model,
                         ROC = TRUE, predict = TRUE, top = 20)
svm.varimp
}
```

Index