Package 'ModelMap'

April 24, 2023

Type Package

Title Modeling and Map Production using Random Forest and Related Stochastic Models

Version 3.4.0.4

Date 2023-04-04

Depends R (>= 2.13.0), randomForest, raster

Suggests party, quantregForest

Imports graphics,grDevices,stats,utils,mgcv,corrplot,fields,HandTill2001,PresenceAbsence

Author Elizabeth Freeman, Tracey Frescino

Maintainer Elizabeth Freeman <elizabeth.a.freeman@usda.gov>

Description Creates sophisticated models of training data and validates the models with an independent test set, cross validation, or Out Of Bag (OOB) predictions on the training data. Create graphs and tables of the model validation results. Applies these models to GIS .img files of predictors to create detailed prediction surfaces. Handles large predictor files for map making, by reading in the .img files in chunks, and output to the .txt file the prediction for each data chunk, before reading the next chunk of data.

License Unlimited

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation no

Repository CRAN

Date/Publication 2023-04-24 20:10:02 UTC

R topics documented:

`ModelMap-package` *Modeling and Map Production using Random Forest and Related Stochastic Models*

Description

Creates sophisticated models of training data and validates the models with an independent test set, cross validation, or with Out Of Bag (OOB) predictions on the training data. Create graphs and tables of the model validation results. Applies these models to GIS .img files of predictors to create detailed prediction surfaces. Handles large predictor files for map making, by reading in the .img files in chunks, and output to the .txt file the prediction for each data chunk, before reading the next chunk of data.

Details

Package:	ModelMap
Type:	Package
Version:	3.4.0.4
Date:	2023-04-04
License:	Unlimited. This code was written and prepared by a U.S. Government employee on official time, and therefore it is

This package provides a push button approach to complex model building and production mapping. It contains three main functions: `model.build`,`model.diagnostics`, and `model.mapmake`.

In addition it contains a simple function `get.test` that can be used to randomly divide a training dataset into training and test/validation sets; `build.rastLUT` that uses GUI prompts to walk a user through the process of setting up a Raster look up table to link predictors from the training data with the rasters used for map contruction; `model.explore`, for preliminary data exploration; and, `model.importance.plot` and `model.interaction.plot` for interpreting the effects of individual model predictors.

`ModelMap` can be run in a traditional R command mode, where all arguments are specified in the function call. However it can also be used in a full push button mode, where you type in the simple command such as model.build, and GUI pop-up windows ask questions about the type of model, the file locations of the data, etc...

Random Forest is implemented through the `randomForest` package within `R`. Random Forest is more user friendly than Stochastic Gradient Boosting, as it has fewer parameters to be set by the user, and is less sensitive to tuning of these parameters. A Random Forest model consists of multiple trees that vote on predictions. For each tree a random subset of the training data is used to construct the tree, with the remaining data points used to construct out-of-bag (OOB) error estimates. At

each node of the tree a random selection of predictors is chosen to determine the split. The number of predictors used to select the splits is the primary user specified parameter that can affect model performance, and this parameter can be automatically optimized using the `randomForest` function `tuneRF()`. Random Forest will not over fit data, therefore the only penalty of increasing the number of trees is computation time. Random Forest can compute variable importance, an advantage over some "black box" modeling techniques if it is important to understand the ecological relationships underlying a model (Brieman, 2001).

Quantile Regression Forests is implemented through the `quantregForest` package.

Conditional Forests is implemented with the `cforest()` function in the `party` package. As stated in the `party` package, ensembles of conditional inference trees have not yet been extensively tested, so this routine is meant for the expert user only and its current state is rather experimental.

For Presence-Absence data, the package `PresenceAbsence` is used for model validation.

For model diagnostics the package `corrplot` is used to plot the correlation between predictor variables.

For map making, the `raster` is used to read and write `.img` files.

For interaction plots, the `fields` package is used to produce image plots.

Author(s)

Author: Elizabeth Freeman and Tracey Frescino

Maintainer: Elizabeth Freeman <elizabeth.a.freeman@usda.gov>

References

Breiman, L. (2001) Random Forests. Machine Learning, 45:5-32.

Elith, J., Leathwick, J. R. and Hastie, T. (2008). A working guide to boosted regression trees. Journal of Animal Ecology. 77:802-813.

Friedman, J.H. (2001). Greedy function approximation: a gradient boosting machine. Ann. Stat., 29(5):1189-1232.

Friedman, J.H. (2002). Stochastic gradient boosting. Comput. Stat. Data An., 38(4):367-378.

Liaw, A. and Wiener, M. (2002). Classification and Regression by randomForest. R News 2(3), 18–22.

N. Meinshausen (2006) "Quantile Regression Forests", Journal of Machine Learning Research 7, 983-999 http://jmlr.csail.mit.edu/papers/v7/

Ridgeway, G., (1999). The state of boosting. Comp. Sci. Stat. 31:172-181

Carolin Strobl, Anne-Laure Boulesteix, Achim Zeileis and Torsten Hothorn (2007). Bias in Random Forest variable Importance Measures: Illustrations, Sources and a Solution. BMC Bioinformatics, 8, 25. http://www.biomedcentral.co,/1471-2105/8/25

Carolin Strobl, James Malley and Gerhard Tutz (2009). An Introduction to Recursive Partitioning: Rationale, Application, and Characteristics of Classification and Regression Trees, Bagging, and Random forests. Phsycological Methods, 14(4), 323-348.

Torsten Hothorn, Berthold Lausen, Axel Benner and Martin Radespiel-Troeger (2004). Bagging Survival Trees. Statistics in Medicine, 23(1), 77-91.

Torsten Hothorn, Peter Buhlmann, Sandrine Dudoit, Annette Molinaro and Mark J. ven der Laan (2006a). Survival Ensembles. Biostatistics, 7(3), 355-373.

Torston Hothorn, Kurt Hornik and Achim Zeileis (2006b). Unbiased Recursive Partitioning: A Conditional Inference Framework. JOurnal of Computational and Graphical Statistics, 15(3), 651-674. Preprint available from http://statmath.wu-wein.ac.at/~zeileis/papers/Hothorn+Hornik+Zeileis-2006.pdf

---

`build.rastLUT` *Build a raster Look-UP-Table for training dataset*

Description

GUI prompts will help the user build a Look-Up-Table to associated predictor variable with their corresponding spatial rasters.

Usage

```
build.rastLUT(imageList=NULL,predList=NULL,qdata.trainfn=NULL,
rastLUTfn=NULL,folder=NULL)
```

Arguments

`imageList`	Vector. A vector of character strings giving names and full paths to all raster data files used in model.
`predList`	Vector. A vector of character strings giving the predictor names used as headers in the model training data.
`qdata.trainfn`	String. The name (full path or base name with path specified by `folder`) of the training data file used for building the model. The file must be a comma-delimited file `*.csv` with column headings. `qdata.trainfn` can also be an `R` dataframe. The column headers from `qdata.trainfn` are used to generate a list of possible predictors for the raster Look-UP-Table.
`rastLUTfn`	String. The name of the file output for the Look-Up-Table. By default, if a file name is provided by the `"qdatatrainfn"` argument `"_rastLUT.csv"` appended after `"qdatatrainfn"`. Otherwise, default filename for look-up-table is`"rastLUT.csv"`
`folder`	String. The folder used for output. Do not add ending slash to path string. If `folder = NULL` (default), a GUI interface prompts user to browse to a folder. To use the working directory, specify `folder = getwd()`.

Details

This function helps the user create a raster Look-Up-Table to be used later by `model.mapmake()`. Currently this function only works in a Windows environment.

First, if `"folder"` is not given, the user selects the output folder for the Look-UP-Table.

Second, if `"predList"` or `"qdatatrainfn"` are not given, the user selects the file containing the training data. The header of the file is used to generate a selection list of possible predictor variables.

Third, if "imageList" is not provided, the user selects the rasters.

Finally, the function steps through each band of each raster, and the user selects the appropriate predictor.

Value

Returns a data frame containing the raster Look-Up-Table. Also Writes a .csv file containing the raster Look-Up-Table.

Author(s)

Elizabeth Freeman

Examples

```
## Not run:
folder<-system.file("extdata", "helpexamples", package = "ModelMap")
qdata.trainfn = paste(folder,"/DATATRAIN.csv",sep="")

#build.rastLUT( qdata.trainfn=qdata.trainfn,
# folder=folder)

## End(Not run) # end dontrun
```

---

col2trans *colors to transparent colors*

Description

transform color names to transparent versions of rgb color codes

Usage

```
col2trans(col.names,alpha=0.5)
```

Arguments

col.names	Vector. Vector of color names from colors.
alpha	Number. Number between 0 and 1 giving alpha channel (opacity) value

Details

Translates a vector of color names to a vector of transparent rgb color codes. Color names must be from names given by colors.

Value

Outputs a vector of transparent color codes.

Author(s)

Elizabeth Freeman

Examples

```
col.names=c("blue","violetred4","thistle3","yellowgreen")
col2trans(col.names,alpha=.2)


###to see effect of alpha###

alpha<-(0:10)/10
colmat<-matrix( 1:(length(alpha)*length(col.names)),
nrow=length(alpha),
ncol=length(col.names),
byrow=TRUE)

color.codes<-vector("character",0)

for(i in 1:length(alpha)){
color.codes<-c(color.codes,col2trans(col.names,alpha=alpha[i]))
}

#make plot#
plot( c(0,1),c(0,1),
type="n",xlab="alpha",ylab="color name",yaxt="n",xaxs="i",yaxs="i")
abline(h=(0:100)/100)
image( z=colmat,
x=(0:length(alpha))/length(alpha),
y=(0:length(col.names))/length(col.names),
col=color.codes,
add=TRUE
)
op<-par(xpd=TRUE)
text( col.names,
x=-.08,
y=(1:length(col.names)-.5)/length(col.names),
srt=90)
par(op)
```

get.test *Randomly Divide Data into Training and Test Sets*

Description

Uses random selection to split a dataset into training and test data sets

Usage

```
get.test(proportion.test, qdatafn = NULL, seed = NULL, folder=NULL,
qdata.trainfn = paste(strsplit(qdatafn, split = ".csv")[[1]], "_train.csv", sep = ""),
qdata.testfn = paste(strsplit(qdatafn, split = ".csv")[[1]], "_test.csv", sep = ""))
```

Arguments

proportion.test	Number. The proportion of the training data that will be randomly extracted for use as a test set. Value between 0 and 1.
qdatafn	String. The name (basename or full path) of the data file to be split into training and test data. This data should include both response and predictor variables. The file must be a comma-delimited file (*.csv) with column headings and the predictor names in the file must match the raster layer files, if applying predictions (predict = TRUE). If NULL (the default), a GUI interface prompts user to browse to the data file.
seed	Integer. The number used to initialize randomization to randomly select rows for a test data set. If you want to produce the same model later, use the same seed. If seed = NULL (the default), a new one is created each time.
folder	String. The folder used for all output from predictions and/or maps. Do not add ending slash to path string. If folder = NULL (default), a GUI interface prompts user to browse to a folder. To use the working directory, specify folder = getwd().
qdata.trainfn	String. The name of the file output of training data. By default, _train appended after qdatafn.
qdata.testfn	String. The name of the file output of test data. By default, _test appended after qdatafn.

Details

This function should be run once, before starting analysis to create training and test sets. If the cross validation option is to be used with RF or SGB models, or if the OOB option is to be used for RF models, then this step is unnecessary.

Value

Outputs a training data file and test data file. Unless qdata.trainfn or qdata.testfn are specified, the output will be located in folder. The output will have the same rows and columns as the original data.

Author(s)

Elizabeth Freeman

Examples

```
## Not run:
qdatafn<-system.file("extdata", "helpexamples","DATATRAIN.csv", package = "ModelMap")

qdata<-read.table(file=qdatafn,sep=",",header=TRUE,check.names=FALSE)

get.test( proportion.test=0.2,
qdatafn=qdatafn,
seed=42,
folder=getwd(),
qdata.trainfn="example.train.csv",
qdata.testfn="example.test.csv")

## End(Not run) # end dontrun
```

`model.build` *Model Building*

Description

Create sophisticated models using Random Forest, Quantile Regression Forests, Conditional Forests, or Stochastic Gradient Boosting from training data

Usage

```
model.build(model.type = NULL, qdata.trainfn = NULL, folder = NULL,
MODELfn = NULL, predList = NULL, predFactor = FALSE, response.name = NULL,
response.type = NULL, unique.rowname = NULL, seed = NULL, na.action = NULL,
keep.data = TRUE, ntree = switch(model.type,RF=500,QRF=1000,CF=500,500),
mtry = switch(model.type,RF=NULL,QRF=ceiling(length(predList)/3),
CF = min(5,length(predList)-1),NULL), replace = TRUE, strata = NULL,
sampsize = NULL, proximity = FALSE, importance=FALSE,
quantiles=c(0.1,0.5,0.9), subset = NULL, weights = NULL,
controls = NULL, xtrafo = NULL, ytrafo = NULL, scores = NULL)
```

Arguments

`model.type`	String. Model type. `"RF"` (random forest), `"QRF"` (quantile random forest), or `"CF"` (conditional forest). The `ModelMap` package does not currently support `SGB` models.

qdata.trainfn — String. The name (full path or base name with path specified by folder) of the training data file used for building the model (file should include columns for both response and predictor variables). The file must be a comma-delimited file *.csv with column headings. qdata.trainfn can also be an R dataframe. If predictions will be made (predict = TRUE or map=TRUE) the predictor column headers must match the names of the raster layer files, or a rastLUT must be provided to match predictor columns to the appropriate raster and band. If qdata.trainfn = NULL (the default), a GUI interface prompts user to browse to the training data file.

folder — String. The folder used for all output from predictions and/or maps. Do not add ending slash to path string. If folder = NULL (default), a GUI interface prompts user to browse to a folder. To use the working directory, specify folder = getwd().

MODELfn — String. The file name to use to save files related to the model object. If MODELfn = NULL (the default), a default name is generated by pasting model.type, response.type, and response.name, separated by underscores. If the other output filenames are left unspecified, MODELfn will be used as the basic name to generate other output filenames. The filename can be the full path, or it can be the simple basename, in which case the output will be to the folder specified by folder.

predList — String. A character vector of the predictor short names used to build the model. These names must match the column names in the training/test data files and the names in column two of the rastLUT. If predList = NULL (the default), a GUI interface prompts user to select predictors from column 2 of rastLUT.

If both predList = NULL and rastLUT = NULL, then a GUI interface prompts user to browse to rasters used as predictors, and select from a generated list, the individual layers (bands) of rasters used to build the model. In this case (i.e., rastLUT = NULL), predictor column names of training data must be standard format, consisting of raster stack name followed by b1, b2, etc..., giving the band number within each stack (Example: stacknameb1, stacknameb2, stacknameb3, ...).

predFactor — String. A character vector of predictor short names of the predictors from predList that are factors (i.e categorical predictors). These must be a subset of the predictor names given in predList Categorical predictors may have multiple categories.

response.name — String. The name of the response variable used to build the model. If response.name = NULL, a GUI interface prompts user to select a variable from the list of column names from training data file. response.name must be column name from the training/test data files.

response.type — String. Response type: "binary", "categorical" or "continuous". Binary response must be binary 0/1 variable with only 2 categories. All zeros will be treated as one category, and everything else will be treated as the second category.

unique.rowname — String. The name of the unique identifier used to identify each row in the training data. If unique.rowname = NULL, a GUI interface prompts user to select a variable from the list of column names from the training data file. If unique.rowname = FALSE, a variable is generated of numbers from 1 to nrow(qdata) to index each row.

seed	Integer. The number used to initialize randomization to build RF or SGB models. If you want to produce the same model later, use the same seed. If seed = NULL (the default), a new seed is created each run.
na.action	String. Model validation. Specifies the action to take if there are NA values in the predictor data. There are 2 options: (1) na.action = na.omit where any data point with missing predictors is removed from the model building data; (2) na.action = na.roughfix where a missing categorical predictor is replaced with the most common category, and a missing continuous predictor or response is replaced with the median. Note: it is not recommended that na.roughfix will just be used for missing predictor. Data points with missing response will always be omitted.
keep.data	Logical. RF and SGB models. Should a copy of the predictor data be included in the model object. Useful for if model.interaction.plot will be used later.
ntree	Integer. RF QRF and CF models. The number of random forest trees for a RF model. The default is 500 trees.
mtry	Integer. RF QRF and CF models. Number of variables to try at each node of Random Forest trees. By default, RF models will use the "tuneRF()" function to optimize mtry.
replace	Logical. RF models. Should sampling of cases be done with or without replacement?
strata	Factor or String. RF models. A (factor) variable that is used for stratified sampling. Can be in the form of either the name of the column in qdata or a factor or vector with one element for each row of qdata.
sampsize	Vector. RF models. Size(s) of sample to draw. For classification, if sampsize is a vector of the length the number of factor levels strata, then sampling is stratified by strata, and the elements of sampsize indicate the numbers to be drawn from each strata. If argument strata is not provided, and repsonse.type = "binary" then sampling is stratified by presence/absence. If argument sampsize is not provided model.build() will use the default value from the randomForest package: if (replace) nrow(data) else ceiling(.632*nrow(data)).
proximity	Logical. RF models. Should proximity measure among the rows be calculated for unsupervised models?
importance	Logical. QRF models. For QRF models only, importance must be specified at the time of model building. If TRUE importance of predictors is assessed at the given quantiles. Warning, on large datasets calculating QRF importances is very memory intensive and may require increasing memory limits with memory.limit(). NOTE: Importance currently unavailable for QRF models.
quantiles	Numeric. Used for QRF models if importance=TRUE. Specify which quantiles of response variable to use. Later importance plots can only be made for quantiles specified at the time of model building.
subset	CF models. An optional vector specifying a subset of observations to be used in the fitting process. Note: subset is not supported for cross validation diagnostics.
weights	CF models. An optional vector of weights to be used in the fitting process. Non-negative integer valued weights are allowed as well as non-negative real

weights. Observations are sampled (with or without replacement) according to probabilities weights/sum(weights). The fraction of observations to be sampled (without replacement) is computed based on the sum of the weights if all weights are integer-valued and based on the number of weights greater zero else. Alternatively, weights can be a double matrix defining case weights for all ncol(weights) trees in the forest directly. This requires more storage but gives the user more control. Note: weights is not supported for cross validation diagnostics.

controls — CF models. An object of class ForestControl-class, which can be obtained using cforest_control (and its convenience interfaces cforest_unbiased and cforest_classical). If controls is specified, then stand alone arguments mtry and ntree ignored and these parameters must be specified as part of the controls argument. If controls not specified, model.build defaults to cforest_unbiased(mtry=mtry, ntree=ntree) with the values of mtry and ntree specified by the stand alone arguments.

xtrafo — CF models. A function to be applied to all input variables. By default, the ptrafo function from the party package is applied. Defaults to xtrafo=ptrafo.

ytrafo — CF models. A function to be applied to all response variables. By default, the ptrafo function from the party package is applied. Defaults to ytrafo=ptrafo.

scores — CF models. An optional named list of scores to be attached to ordered factors. Note: weights is not supported for cross validation diagnostics.

Details

This package provides a push button approach to complex model building and production mapping. It contains three main functions: model.build,model.diagnostics, and model.mapmake.

In addition it contains a simple function get.test that can be used to randomly divide a training dataset into training and test/validation sets; build.rastLUT that uses GUI prompts to walk a user through the process of setting up a Raster look up table to link predictors from the training data with the rasters used for map contruction; model.explore, for preliminary data exploration; and, model.importance.plot and model.interaction.plot for interpreting the effects of individual model predictors.

These functions can be run in a traditional R command mode, where all arguments are specified in the function call. However they can also be used in a full push button mode, where you type in, for example, the simple command model.build, and GUI pop up windows will ask questions about the type of model, the file locations of the data, etc...

When running the ModelMap package on non-Windows platforms, file names and folders need to be specified in the argument list, but other pushbutton selections are handled by the select.list() function, which is platform independent.

Binary, categorical, and continuous response models are supported for Random Forest and Conditional Forest. Quantile Random Forest is appropriate for only continuous response models.

Random Forest is implemented through the randomForest package within R. Random Forest is more user friendly than Stochastic Gradient Boosting, as it has fewer parameters to be set by the user, and is less sensitive to tuning of these parameters. A Random Forest model consists of multiple trees that vote on predictions. For each tree a random subset of the training data is used to construct the tree, with the remaining data points used to construct out-of-bag (OOB) error estimates. At each

node of the tree a random selection of predictors is chosen to determine the split. The number of predictors used to select the splits (argument `mtry`) is the primary user specified parameter that can affect model performance.

By default `mtry` will be automatically optimized using the `randomForest` package `tuneRF()` function. Note that this is a stochastic process. If there is a chance that models may be combined later with the `randomForest` package `combine` function then for consistency it is important to provide the `mtry` argument rather that using the default optimization process.

Random Forest will not over fit data, therefore the only penalty of increasing the number of trees is computation time. Random Forest can compute variable importance, an advantage over some "black box" modeling techniques if it is important to understand the ecological relationships underlying a model (Brieman, 2001).

Quantile Regression Forests is implemented through the `quantregForest` package.

Conditional Forests is implemented with the `cforest()` function in the `party` package. As stated in the `party` package, ensembles of conditional inference trees have not yet been extensively tested, so this routine is meant for the expert user only and its current state is rather experimental.

For CF models, `ModelMap` currently only supports binary, categorical and continuous response models. Also, for some CF model parameters (`subset`, `weights`, and `scores`) `ModelMap` only provides OOB and independent test set diagnostics, and does not support cross validation diagnostics.

Stochastic gradient boosting is not currently supported by `ModelMap`.

Value

The function will return the model object. Additionally, it will write a text file to disk, in the folder specified by `folder`. This file lists the values of each argument as chosen from GUI prompts used for the function call.

Author(s)

Elizabeth Freeman and Tracey Frescino

References

Breiman, L. (2001) Random Forests. Machine Learning, 45:5-32.

Elith, J., Leathwick, J. R. and Hastie, T. (2008). A working guide to boosted regression trees. Journal of Animal Ecology. 77:802-813.

Liaw, A. and Wiener, M. (2002). Classification and Regression by randomForest. R News 2(3), 18–22.

N. Meinshausen (2006) "Quantile Regression Forests", Journal of Machine Learning Research 7, 983-999 http://jmlr.csail.mit.edu/papers/v7/

Ridgeway, G., (1999). The state of boosting. Comp. Sci. Stat. 31:172-181

Carolin Strobl, Anne-Laure Boulesteix, Achim Zeileis and Torsten Hothorn (2007). Bias in Random Forest variable Importance Measures: Illustrations, Sources and a Solution. BMC Bioinformatics, 8, 25. http://www.biomedcentral.co,/1471-2105/8/25

Carolin Strobl, James Malley and Gerhard Tutz (2009). An Introduction to Recursive Partitioning: Rationale, Application, and Characteristics of Classification and Regression Trees, Bagging, and Random forests. Phsycological Methods, 14(4), 323-348.

Torsten Hothorn, Berthold Lausen, Axel Benner and Martin Radespiel-Troeger (2004). Bagging Survival Trees. Statistics in Medicine, 23(1), 77-91.

Torsten Hothorn, Peter Buhlmann, Sandrine Dudoit, Annette Molinaro and Mark J. ven der Laan (2006a). Survival Ensembles. Biostatistics, 7(3), 355-373.

Torston Hothorn, Kurt Hornik and Achim Zeileis (2006b). Unbiased Recursive Partitioning: A Conditional Inference Framework. Journal of Computational and Graphical Statistics, 15(3), 651-674. Preprint available from http://statmath.wu-wein.ac.at/~zeileis/papers/Hothorn+Hornik+Zeileis-2006.pdf

See Also

get.test, model.diagnostics, model.mapmake

Examples

```
## Not run:
################################################################################
############################## Run this set up code: ###########################
################################################################################

# set seed:
seed=38

# Define training and test files:

qdata.trainfn = system.file("extdata", "helpexamples","DATATRAIN.csv", package = "ModelMap")

# Define folder for all output:
folder=getwd()

#identifier for individual training and test data points

unique.rowname="ID"


################################################################################
############## Pick one of the following sets of definitions: ##############
################################################################################


########## Continuous Response, Continuous Predictors ############

#file name:
MODELfn="RF_Bio_TC"

#predictors:
predList=c("TCB","TCG","TCW")

#define which predictors are categorical:
predFactor=FALSE
```

```
# Response name and type:
response.name="BIO"
response.type="continuous"


########## binary Response, Continuous Predictors ############

#file name to store model:
MODELfn="RF_CONIFTYP_TC"

#predictors:
predList=c("TCB","TCG","TCW")

#define which predictors are categorical:
predFactor=FALSE

# Response name and type:
response.name="CONIFTYP"

# This variable is 1 if a conifer or mixed conifer type is present,
# otherwise 0.

response.type="binary"


########## Continuous Response, Categorical Predictors ############

# In this example, NLCD is a categorical predictor.
#
# You must decide what you want to happen if there are categories
# present in the data to be predicted (either the validation/test set
# or in the image file) that were not present in the original training data.
# Choices:
#       na.action = "na.omit"
#                   Any validation datapoint or image pixel with a value for any
#                   categorical predictor not found in the training data will be
#                   returned as NA.
#       na.action = "na.roughfix"
#                   Any validation datapoint or image pixel with a value for any
#                   categorical predictor not found in the training data will have
#                   the most common category for that predictor substituted,
#                   and the a prediction will be made.

# You must also let R know which of the predictors are categorical, in other
# words, which ones R needs to treat as factors.
# This vector must be a subset of the predictors given in predList

#file name to store model:
MODELfn="RF_BIO_TCandNLCD"

#predictors:
predList=c("TCB","TCG","TCW","NLCD")
```

```
#define which predictors are categorical:
predFactor=c("NLCD")

# Response name and type:
response.name="BIO"
response.type="continuous"



##############################################################################
########################### build model: #####################################
##############################################################################


### create model before batching (only run this code once ever!) ###

model.obj = model.build( model.type="RF",
                        qdata.trainfn=qdata.trainfn,
                        folder=folder,
                        unique.rowname=unique.rowname,
                        MODELfn=MODELfn,
                        predList=predList,
                        predFactor=predFactor,
                        response.name=response.name,
                        response.type=response.type,
                        seed=seed,
                        na.action="na.roughfix"
)


## End(Not run) # end dontrun
```

`model.diagnostics` *Model Predictions and Diagnostics*

Description

Takes model object and makes predictions, runs model diagnostics, and creates graphs and tables of the results.

Usage

```
model.diagnostics(model.obj = NULL, qdata.trainfn = NULL, qdata.testfn = NULL,
folder = NULL, MODELfn = NULL, response.name = NULL, unique.rowname = NULL,
 diagnostic.flag=NULL, seed = NULL, prediction.type=NULL, MODELpredfn = NULL,
 na.action = NULL, v.fold = 10, device.type = NULL, DIAGNOSTICfn = NULL,
 res=NULL, jpeg.res = 72, device.width = 7,  device.height = 7, units="in",
 pointsize=12, cex=par()$cex, req.sens, req.spec, FPC, FNC, quantiles=NULL,
 all=TRUE, subset = NULL, weights = NULL, mtry = NULL, controls = NULL,
 xtrafo = NULL, ytrafo = NULL, scores = NULL)
```

Arguments

model.obj	R model object. The model object to use for prediction. The model object must be of type "RF" (random forest), "QRF" (quantile random forest), or "CF" (conditional forest). The ModelMap package does not currently support SGB models.
qdata.trainfn	String. The name (full path or base name with path specified by folder) of the training data file used for building the model (file should include columns for both response and predictor variables). The file must be a comma-delimited file *.csv with column headings. qdata.trainfn can also be an R dataframe. If predictions will be made (predict = TRUE or map=TRUE) the predictor column headers must match the names of the raster layer files, or a rastLUT must be provided to match predictor columns to the appropriate raster and band. If qdata.trainfn = NULL (the default), a GUI interface prompts user to browse to the training data file.
qdata.testfn	String. The name (full path or base name with path specified by folder) of the independent data set for testing (validating) the model's predictions. The file must be a comma-delimited file ".csv" with column headings and the column headings must be the same as those in the training data file. qdata.testfn can also be an R dataframe. If qdata.testfn = NULL (default), a GUI interface asks user if there is a test set available, then prompts user to browse to the test data file. If no test set is desired (for example, cross-fold validation will be performed, or for RF models, Out-Of-Bag estimation, set qdata.testfn = FALSE. If no test set is given, and qdata.testfn is not set to FALSE, the GUI interface asks if a proportion of the data should be set aside as an independent test set. If this is desired, the user will be prompted to specify the proportion to set aside as test data, and two new data files will be generated in the out put folder. The new file names will be the original data file name with "_train" and "_test" appended to the end of the file names.
folder	String. The folder used for all output from predictions and/or maps. Do not add ending slash to path string. If folder = NULL (default), a GUI interface prompts user to browse to a folder. To use the working directory, specify folder = getwd().
MODELfn	String. The file name to use to save the generated model object. If MODELfn = NULL (the default), a default name is generated by pasting model.type_response.type_response.name. If the other output filenames are left unspecified, MODELfn will be used as the basic name to generate other output filenames. The filename can be the full path, or it can be the simple basename, in which case the output will be to the folder specified by folder.
response.name	String. The name of the response variable used to build the model. The response.name must be column name from the training/test data files. If the model.obj was constructed in ModelMap with the model.build() function, then the model.diagnostics() can extract the response.name from the model.obj. If the model was constructed outside of ModelMap the you may need to specify the response.name. In particular, if a SGB model was constructed with the aid of Elith's code, it is necessary to specify the response.name argument, as all models constructed with this code are given a response name of "y.data". If the response.name argument differs from the response name in the model.obj, the specified argument is giver preference, and a warning generated.

`unique.rowname` String. The name of the unique identifier used to identify each row in the training data. If `unique.rowname = NULL`, a GUI interface prompts user to select a variable from the list of column names from the training data file. If `unique.rowname = FALSE`, a variable is generated of numbers from 1 to `nrow(qdata)` to index each row.

`diagnostic.flag`
String. The name of a column used to identify a subset of rows in the training data or test data to use for model diagnostics. This column must be either a logical vector (`TRUE` and `FALSE`) or a vector of zeros ond ones (where `0=FALSE` and `1=TRUE`. If this argument is used model diagnostics that depend on predicted and observed values will be calculated from a subset of the training or test data. These include confusion matrix and threshold criteria for binary response models and the scatterplot for continuous response models. The output file of predicted and observed values will have an aditional column, indicating which rows were used in the diagnostic calculations. Note that for cross validation, the entire training dataset will be used to create cross validation predictions, but that only the predictions on the the rows indicated by `diagnostic.flag` will be used for the diagnostics.

`seed` Integer. The number used to initialize randomization to build RF or SGB models. If you want to produce the same model later, use the same seed. If `seed = NULL` (the default), a new seed is created each run.

`prediction.type`
String. Prediction type. `"TEST"`, `"CV"`, `"OOB"` or `"TRAIN"`. If `predict = "TEST"`, validation predictions will be made on the test set provided by `qdata.testfn`. If `predict = "CV"`, cross validation will be used on the training data provided by `qdata.trainfn`. If `model.obj` is a Random Forest model and `predict = "OOB"` the Out-of-Bag predictions will be calculated on the training data. If `model.obj` is a Stochastic Gradient Boosting model and `predict = "TRAIN"` the predictions will be calculated on the training data, but these predictions should be used with caution as this will lead to over optimistic estimates of model quality. A `*.csv` file of the unique id, observed, and predicted values is generated and put in the specified (or default) folder.

`MODELpredfn` String. Model validation. A character string used to construct the output file names for the validation diagnostics, for example the prediction `*.csv` file, and the graphics `*.jpg`, `*.pdf` and `*.ps` files. The filename can be the full path, or it can be the simple basename, in which case the output will be to the folder specified by `folder`. If `MODELpredfn = NULL` (the default), a default name is created by pasting `modelfn` and `"_pred"`.

`na.action` String. Model validation. Specifies the action to take if there are `NA` values in the predictor data or if there is a level or class of a categorical predictor variable in the validation test set, but not in the training data set. By default, `model.daignostics()` will use the same `na.action` as was given to `model.build`. There are 2 options: (1) `na.action = "na.omit"` where any data point with `NA` or any new levels for any of the factored predictors is removed from the data; (2) `na.action = "na.roughfix"` where a missing categorical predictor is replaced with the most common category, and a missing continuous predictor is replaced with the median. Note: data points with missing response values will always be omitted.

v.fold — Integer (or logical FALSE). Model validation. The number of cross validation folds to use when making validation predictions on the training data. Only used if prediction.type = "CV".

device.type — String or vector of strings. Model validation. One or more device types for graphical output from model validation diagnostics.

Current choices:

"default"	default graphics device
"jpeg"	*.jpg files
"none"	no graphics device generated
"pdf"	*.pdf files
"png"	*.png files
"postscript"	*.ps files
"tiff"	*.tif files

DIAGNOSTICfn — String. Model validation. Name used as base to create names for output files from model validation diagnostics. The filename can be the full path, or it can be the simple basename, in which case the output will be to the folder specified by folder. Defaults to DIAGNOSTICfn = MODELfn followed by the appropriate suffixes (i.e. ".csv", ".jpg", etc...).

res — Integer. Model validation. Pixels per inch for jpeg, png, and tiff plots. The default is 72dpi, good for on screen viewing. For printing, suggested setting is 300dpi.

jpeg.res — Integer. Model validation. Deprecated. Ignored unless res not provided.

device.width — Integer. Model validation. The device width for diagnostic plots in inches.

device.height — Integer. Model validation. The device height for diagnostic plots in inches.

units — Model validation. The units in which device.height and device.width are given. Can be "px" (pixels), "in" (inches, the default), "cm" or "mm".

pointsize — Integer. Model validation. The default pointsize of plotted text, interpreted as big points (1/72 inch) at res ppi

cex — Integer. Model validation. The cex for diagnostic plots.

req.sens — Numeric. Model validation. The required sensitivity for threshold optimization for binary response model evaluation.

req.spec — Numeric. Model validation. The required specificity for threshold optimization for binary response model evaluation.

FPC — Numeric. Model validation. The False Positive Cost for threshold optimization for binary response model evaluation.

FNC — Numeric. Model validation. The False Negative Cost for threshold optimization for binary response model evaluation.

quantiles — Numeric Vector. QRF models. The quantiles to predict. A numeric vector with values between zero and one. If model was built without specifying quantiles, quantile importance can not be calculated, but quantiles can still be used to specify prediction quantiles. If model was built with quantiles specified, then the model quantiles will be used for importance graph. If quantiles are not specified for model building or diagnostics, prediction quantiles will default to quantiles=c(0.1,0.5,0.9)

all	Logical. QRF models. all=TRUE uses all observations for prediction. all=FALSE uses only a certain number of observations per node for prediction (set with argument obs). Unlike in the quantredForest package itself, the default in ModelMap is all=TRUE, to more closely parallel ordinary random forest models.
subset	CF models. NOT SUPPORTED. Only needed for prediction.type="CV" for CF models. An optional vector specifying a subset of observations to be used in the fitting process. Note: subset is not yet supported for cross validation diagnostics.
weights	CF models. NOT SUPPORTED. Only needed for prediction.type="CV" for CF models. An optional vector of weights to be used in the fitting process. Non-negative integer valued weights are allowed as well as non-negative real weights. Observations are sampled (with or without replacement) according to probabilities weights/sum(weights). The fraction of observations to be sampled (without replacement) is computed based on the sum of the weights if all weights are integer-valued and based on the number of weights greater zero else. Alternatively, weights can be a double matrix defining case weights for all ncol(weights) trees in the forest directly. This requires more storage but gives the user more control. Note: weights is not yet supported for cross validation diagnostics.
mtry	Integer. Only needed for prediction.type="CV" for CF models (for RF and QRF models mtry will be determined from the model object). Number of variables to try at each node of Random Forest trees.
controls	CF models. Only needed for prediction.type="CV" for CF models. An object of class ForestControl-class, which can be obtained using cforest_control (and its convenience interfaces cforest_unbiased and cforest_classical). If controls is specified, then stand alone arguments mtry and ntree ignored and these parameters must be specified as part of the controls argument. If controls not specified, model.build defaults to cforest_unbiased(mtry=mtry, ntree=ntree) with the values of mtry and ntree specified by the stand alone arguments.
xtrafo	CF models. Only needed for prediction.type="CV" for CF models. A function to be applied to all input variables. By default, the ptrafo function from the party package is applied.
ytrafo	CF models. Only needed for prediction.type="CV" for CF models. A function to be applied to all response variables. By default, the ptrafo function from the party package is applied.
scores	CF models. NOT SUPPORTED. Only needed for prediction.type="CV" for CF models. An optional named list of scores to be attached to ordered factors. Note: scores is not yet supported for cross validation diagnostics.

Details

model.diagnostics()takes model object and makes predictions, runs model diagnostics, and creates graphs and tables of the results.

model.diagnostics() can be run in a traditional R command mode, where all arguments are specified in the function call. However it can also be used in a full push button mode, where you type in the simple command model.map(), and GUI pop up windows will ask questions about the type of model, the file locations of the data, etc...

When running `model.map()` on non-Windows platforms, file names and folders need to be specified in the argument list, but other pushbutton selections are handled by the `select.list()` function, which is platform independent.

Diagnostic predictions are made my one of four methods, and a text file is generated consisting of three columns: Observation ID, observed values and predicted values. If `predition.type = "CV")` an additional column indicates which cross-fold each observation fell into. If the models response type is categorical then in addition a column giving the category predicted by majority vote, there are also categories for each possible response category giving the proportion of trees that predicted that category.

A variable importance graph is made. If `response.type = "categorical"`, category specific graphs are generated for variable importance. These show how much the model accuracy for each category is affected when the values of each predictor variable is randomly permuted.

The package `corrplot` is used to generate a plot of correlation between predictor variables. If there are highly correlated predictor variables, then the variable importances of `"RF"` and `"QRF"` models need to be interpreted with care, and users may want to consider looking at the conditional variable importances available for `"CF"` models produced by the `party` package.

If `model.type = "RF"`, the OOB error is plotted as a function of number of trees in the model. If `response.type = "binary"` or If `response.type = "categorical"` category specific graphs are generated for OOB error as a function of number of trees.

If `response.type = "binary"`, a summary graph is made using the `PresenceAbsence` package and a `*.csv` spreadsheets are created of optimized thresholds by several methods with their associated error statistics, and predicted prevalence.

If `response.type = "continuous"` a scatterplot of observed vs. predicted is created with a simple linear regression line. The graph is labeled with slope and intercept of this line as well as Pearson's and Spearman's correlation coefficients.

If `response.type = "categorical"`, a confusion matrix is generated, that includes erros of ommission and comission, as well as Kappa, Percent Correctly Classified (PCC) and the Multicategorical Area Under the Curve (MAUC) as defined by Hand and Till (2001) and calculated by the package `HandTill2001`.

Value

The function will return a dataframe of the row ID, and the Observed and predicted values.

For Binary response models the predicted probability of presence is returned.

For Categorical Response models the predicted category (by majority vote) is returned as well as a column for each category giving the probability of that category. If necessary, `make.names` is applied to the categories to create valid column names.

For Continuous response models the predicted value is returned.

If `prediction.type = "CV"` the dataframe also includes a column indicating which cross-validation fold each datapoint was in.

Note

Importance currently unavailable for QRF models.

If you are running cross validation diagnostics on a CF model, the model parameters will NOT automatically be passed to `model.diagnostics()`. For cross validation, it is the users responsibility to be certain that the CF arguments are the same in `model.build()` and `model.diagnostics()`.

Also, for some CF model parameters (`subset`, `weights`, and `scores`) `ModelMap` only provides OOB and independent test set diagnostics, and does not support cross validation diagnostics.

Author(s)

Elizabeth Freeman and Tracey Frescino

References

Breiman, L. (2001) Random Forests. Machine Learning, 45:5-32.

Elith, J., Leathwick, J. R. and Hastie, T. (2008). A working guide to boosted regression trees. Journal of Animal Ecology. 77:802-813.

Hand, D. J., & Till, R. J. (2001). A simple generalisation of the area under the ROC curve for multiple class classification problems. Machine Learning, 45(2), 171-186.

Liaw, A. and Wiener, M. (2002). Classification and Regression by randomForest. R News 2(3), 18–22.

Ridgeway, G., (1999). The state of boosting. Comp. Sci. Stat. 31:172-181

See Also

`get.test`, `model.build`, `model.mapmake`

Examples

```
## Not run:

################################################################################
############################## Run this set up code: ###########################
################################################################################

# set seed:
seed=38

# Define training and test files:

qdata.trainfn = system.file("extdata", "helpexamples","DATATRAIN.csv", package = "ModelMap")
qdata.testfn = system.file("extdata", "helpexamples","DATATEST.csv", package = "ModelMap")

# Define folder for all output:
folder=getwd()

#identifier for individual training and test data points

unique.rowname="ID"
```

```
##########################################################################
############## Pick one of the following sets of definitions: #############
##########################################################################


########## Continuous Response, Continuous Predictors ############

#file name to store model:
MODELfn="RF_Bio_TC"

#predictors:
predList=c("TCB","TCG","TCW")

#define which predictors are categorical:
predFactor=FALSE

# Response name and type:
response.name="BIO"
response.type="continuous"


########## binary Response, Continuous Predictors ############

#file name to store model:
MODELfn="RF_CONIFTYP_TC"

#predictors:
predList=c("TCB","TCG","TCW")

#define which predictors are categorical:
predFactor=FALSE

# Response name and type:
response.name="CONIFTYP"

# This variable is 1 if a conifer or mixed conifer type is present,
# otherwise 0.

response.type="binary"


########## Continuous Response, Categorical Predictors ############

# In this example, NLCD is a categorical predictor.
#
# You must decide what you want to happen if there are categories
# present in the data to be predicted (either the validation/test set
# or in the image file) that were not present in the original training data.
# Choices:
#       na.action =  "na.omit"
#                    Any validation datapoint or image pixel with a value for any
#                    categorical predictor not found in the training data will be
#                    returned as NA.
```

```
#       na.action =  "na.roughfix"
#                    Any validation datapoint or image pixel with a value for any
#                    categorical predictor not found in the training data will have
#                    the most common category for that predictor substituted,
#                    and the a prediction will be made.

# You must also let R know which of the predictors are categorical, in other
# words, which ones R needs to treat as factors.
# This vector must be a subset of the predictors given in predList

#file name to store model:
MODELfn="RF_BIO_TCandNLCD"

#predictors:
predList=c("TCB","TCG","TCW","NLCD")

#define which predictors are categorical:
predFactor=c("NLCD")

# Response name and type:
response.name="BIO"
response.type="continuous"



##############################################################################
########################### build model: #####################################
##############################################################################


### create model ###

model.obj = model.build( model.type="RF",
                      qdata.trainfn=qdata.trainfn,
                      folder=folder,
                      unique.rowname=unique.rowname,
                      MODELfn=MODELfn,
                      predList=predList,
                      predFactor=predFactor,
                      response.name=response.name,
                      response.type=response.type,
                      seed=seed,
                      na.action="na.roughfix"
)

##############################################################################
#### Then Run this code make validation predictions and diagnostics: ######
##############################################################################


### for Out-of-Bag predictions ###

MODELpredfn<-paste(MODELfn,"_OOB",sep="")
```

```
PRED.OOB<-model.diagnostics(  model.obj=model.obj,
qdata.trainfn=qdata.trainfn,
                    folder=folder,
                     unique.rowname=unique.rowname,
                  # Model Validation Arguments
                     prediction.type="OOB",
                     MODELpredfn=MODELpredfn,
                     device.type=c("default","jpeg","pdf"),
                     na.action="na.roughfix"
)
PRED.OOB

### for Cross-Validation predictions ###

#MODELpredfn<-paste(MODELfn,"_CV",sep="")
#PRED.CV<-model.diagnostics(  model.obj=model.obj,
#                     qdata.trainfn=qdata.trainfn,
#                     folder=folder,
#                     unique.rowname=unique.rowname,
#                     seed=seed,
#                  # Model Validation Arguments
#                     prediction.type="CV",
#                     MODELpredfn=MODELpredfn,
#                     device.type=c("default","jpeg","pdf"),
#                     v.fold=10,
#                     na.action="na.roughfix"
#)
#PRED.CV

### for Independent Test Set predictions ###

#MODELpredfn<-paste(MODELfn,"_TEST",sep="")
#PRED.TEST<-model.diagnostics(  model.obj=model.obj,
#                     qdata.testfn=qdata.testfn,
#                     folder=folder,
#                     unique.rowname=unique.rowname,
#                  # Model Validation Arguments
#                     prediction.type="TEST",
#                     MODELpredfn=MODELpredfn,
#                     device.type=c("default","jpeg","pdf"),
#                     na.action="na.roughfix"
#)
#PRED.TEST
)


## End(Not run) # end dontrun
```

`model.explore` *Exploratory data analysis*

Description

Graphically explores the relationships between the training data and the predictor rasters.

Usage

```
model.explore(qdata.trainfn = NULL, folder = NULL, predList = NULL,
predFactor = FALSE, response.name = NULL, response.type = NULL,
response.colors = NULL, unique.rowname = NULL, OUTPUTfn = NULL,
device.type = NULL, allow.default.graphics=FALSE, res=NULL, jpeg.res = 72,
MAXCELL=100000, device.width = NULL, device.height = NULL, units="in",
pointsize=12, cex=1, rastLUTfn = NULL, create.extrapolation.masks = FALSE,
na.value = -9999, col.ramp = rainbow(101, start = 0, end = 0.5),
col.cat = palette()[-1])
```

Arguments

qdata.trainfn	String. The name (full path or base name with path specified by folder) of the training data file used for building the model (file should include columns for both response and predictor variables). The file must be a comma-delimited file *.csv with column headings. qdata.trainfn can also be an R dataframe. If predictions will be made (predict = TRUE or map=TRUE) the predictor column headers must match the names of the raster layer files, or a rastLUT must be provided to match predictor columns to the appropriate raster and band. If qdata.trainfn = NULL (the default), a GUI interface prompts user to browse to the training data file.
folder	String. The folder used for all output from predictions and/or maps. Do not add ending slash to path string. If folder = NULL (default), a GUI interface prompts user to browse to a folder. To use the working directory, specify folder = getwd().
predList	String. A character vector of the predictor short names used to build the model. These names must match the column names in the training/test data files and the names in column two of the rastLUT. If predList = NULL (the default), a GUI interface prompts user to select predictors from column 2 of rastLUT.
predFactor	String. A character vector of predictor short names of the predictors from predList that are factors (i.e categorical predictors). These must be a subset of the predictor names given in predList Categorical predictors may have multiple categories.
response.name	String. The name of the response variable used to build the model. If response.name = NULL, a GUI interface prompts user to select a variable from the list of column names from training data file. response.name must be column name from the training/test data files.
response.type	String. Response type: "binary", "categorical" or "continuous". Binary response must be binary 0/1 variable with only 2 categories. All zeros will be treated as one category, and everything else will be treated as the second category.

response.colors

Data frame. A two column data frame. Column names must be:category, the response categories; and, color, the colors associated with each category.

unique.rowname String. The name of the unique identifier used to identify each row in the training data. If unique.rowname = NULL, a GUI interface prompts user to select a variable from the list of column names from the training data file. If unique.rowname = FALSE, a variable is generated of numbers from 1 to nrow(qdata) to index each row.

OUTPUTfn String. Filename that ouput file names will be based on.

device.type String or vector of strings. Model validation. One or more device types for graphical output from model validation diagnostics.

Current choices:

"default"	default graphics device
"jpeg"	*.jpg files
"none"	no graphics device generated
"pdf"	*.pdf files
"png"	*.png files
"postscript"	*.ps files
"tiff"	*.tif files

Note that the "default" device is disabled unless allow.default.graphics=TRUE. This is because these graphics are slow to produce, and if the onscreen graphics window is moved or closed while the function is in progress there is a risk of crashing the entire R session.

allow.default.graphics

Logical. Should the default on-screen graphics device be allowed. USE WITH CAUTION! These graphics are complicated and slow to produce. If the on-screen default graphics device is moved or closed before the plot is completed it can crash the entire R session.

res Integer. Model validation. Pixels per inch for jpeg, png, and tiff plots. The default is 72dpi, good for on screen viewing. For printing, suggested setting is 300dpi.

jpeg.res Integer. Graphical output. Deprecated. Ignored unless res not provided.

MAXCELL Integer. Graphical output. The maximum number of raster cells used to create the graphical output. Rasters larger than this value will be subsampled for the graphical maps and figures. The default value of MAXCELL=100000 is generally a good resolution for onscreen viewing with the default jpeg resolution of 72dpi. Publication quality qraphics may require higher MAXCELL. Higher values require more memory and are slower to process.

Note: MAXCELL only affects graphical figures. Output rasters generated when create.extrapolation.masks=TRUE are always done on full resolution rasters.

device.width Integer. Model validation. The device width for diagnostic plots in inches.

device.height Integer. Model validation. The device height for diagnostic plots in inches.

units Model validation. The units in which device.height and device.width are given. Can be "px" (pixels), "in" (inches, the default), "cm" or "mm".

`pointsize` Integer. Model validation. The default pointsize of plotted text, interpreted as big points (1/72 inch) at `res` ppi

`cex` Integer. Model validation. The cex for diagnostic plots.

`rastLUTfn` String. The file name (full path or base name with path specified by `folder`) of a `.csv` file for a `rastLUT`. Alternatively, a dataframe containing the same information. The `rastLUT` must include 3 columns: (1) the full path and name of the raster file; (2) the shortname of each predictor / raster layer (band); (3) the layer (band) number. The shortname (column 2) must match the names `predList`, the predictor column names in training/test data set (`qdata.trainfn` and `qdata.testfn`, and the predictor names in `model.obj`.

Example of comma-delimited file:

```
C:/button_test/tc99_2727subset.img,   tc99_2727subsetb1,   1
C:/button_test/tc99_2727subset.img,   tc99_2727subsetb2,   2
C:/button_test/tc99_2727subset.img,   tc99_2727subsetb3,   3
```

`create.extrapolation.masks` Logical. If `TRUE` then the raster brick containing the masks for all predictors from `predList` is saved as image file. The layers in this file will be in the same order as the predictors in `predList`

`na.value` Value used in rasters to indicate `NA`. Note this value is only used for `NA` values in the predictor rasters. Note: all predictor rasters must use the same value for `NA`. `NA` values in the training data should be indicated with `NA`.

`col.ramp` Color ramp to use for continuous predictors

`col.cat` Vector. Vector of colors to use for categorical predictors.

Details

The `model.explore` function is intended to aid with preliminary data exploration before model building. It includes graphical tools to explore the relationships between the training data (both predictors and responses) as well as the predictor rasters. It uses the `corrplot` package to create a correlation plot of the continuous predictor. This can aid in interpreting the `model.importance.plot` output from the models, as Random Forest models divide importance between correlated predictors, while Stochastic Gradient Boosting models assing the majority of the importance to the correlated predictor that is used earlies in the model.

The `model.explore` function also can aid in identifying if additional training data is needed. For example, the maps of the extrapolation masks for the predictor rasters help spot areas of the map where the pixels lie outside the range of the training data, and therefore any model predictions will be extrapolations, and possibly unreliable. The user can decide to either collect additional training data, or mask out these areas of the final prediction output of `model.mapmake`.

To increase speed, the default behavior for large predictor rasters is to create the graphics from subsampled rasters. (Note: for categorical predictors, the full raster is always used to identify all categories found in the map area.) If `create.extrapolation.masks=TRUE`, then the full rasters are used for the extrapolation masks, regardless of size of the reasters. This option runs much slower, as large rasters need to be read into R a block at a time.

Value

Function does not return a value, but does create files.

Graphical files are created for each predictor variable, with file type determined by `device.type`. In addition, if `create.extrapolation.masks`, an extrapolation mask raster is created for each predictor as well as an overall extrapolation mask, with the value `1` for pixels with predictor values within the range of the training data, or categories found in the training data, and the value `0` for pixels outside the range of the training data, categories not found in the training data, or NA value. The overall extrapolation mask has `0` if any of the predictors for that pixel are extrapolated. Note that this option is much slower to run.

Note

The default graphics device is disabled unless `allow.default.graphics` is set to `TRUE`. These graphics can be slow to produce, and if the on screen graphics device is moved or closed while the graphic is in progress, it can crash R. It is recomended that graphics be written to a file by using jpeg, pdf, etc... `device.type`.

Author(s)

Elizabeth Freeman

Examples

```
## Not run:

#############################################################################
############################## Run this set up code: ########################
#############################################################################

###Define training and test files:
qdata.trainfn = system.file("extdata", "helpexamples","DATATRAIN.csv", package = "ModelMap")

###Define folder for all output:
folder=getwd()

###identifier for individual training and test data points
unique.rowname="ID"

###predictors:
predList=c("TCB","TCG","TCW","NLCD")

###define which predictors are categorical:
predFactor=c("NLCD")

###Create a the filename (including path) for the rast Look up Tables ###
rastLUTfn.2001 <- system.file( "extdata",
"helpexamples",
"LUT_2001.csv",
package="ModelMap")
```

```
###Load rast LUT table, and add path to the predictor raster filenames in column 1 ###
rastLUT.2001 <- read.table(rastLUTfn.2001,header=FALSE,sep=",",stringsAsFactors=FALSE)

for(i in 1:nrow(rastLUT.2001)){
rastLUT.2001[i,1] <- system.file("extdata",
"helpexamples",
rastLUT.2001[i,1],
package="ModelMap")
}

##################Continuous Response###################

###Response name and type:
response.name="BIO"
response.type="continuous"

###file name to store model:
OUTPUTfn="BIO_TCandNLCD.img"

###run model.explore

model.explore( qdata.trainfn=qdata.trainfn,
folder=folder,
predList=predList,
predFactor=predFactor,

response.name=response.name,
response.type=response.type,

unique.rowname=unique.rowname,

OUTPUTfn=OUTPUTfn,
device.type="jpeg",
jpeg.res=144,


# Raster arguments
rastLUTfn=rastLUT.2001,
na.value=-9999,

# colors for continuous predictors
col.ramp=rainbow(101,start=0,end=.5),
# colors for categorical predictors
col.cat=c("wheat1","springgreen2","darkolivegreen4",
  "darkolivegreen2","yellow","thistle2",
  "brown2","brown4")
)

## End(Not run) # end dontrun
```

`model.importance.plot` *Compares the variable importance of two models with a back to back barchart.*

Description

Takes two models and produces a back to back bar chart to compare the importance of the predictor variables. Models can be any combination of Random Forest or Stochastic Gradient Boosting, as long as both models have the same predictor variables.

Usage

```
model.importance.plot(model.obj.1 = NULL, model.obj.2 = NULL,
model.name.1 = "Model 1", model.name.2 = "Model 2", imp.type.1 = NULL,
imp.type.2 = NULL, type.label=TRUE, class.1 = NULL, class.2 = NULL,
quantile.1=NULL, quantile.2=NULL,
col.1="grey", col.2="black", scale.by = "sum", sort.by = "model.obj.1",
cf.mincriterion.1 = 0, cf.conditional.1 = FALSE, cf.threshold.1 = 0.2,
cf.nperm.1 = 1, cf.mincriterion.2 = 0, cf.conditional.2 = FALSE,
cf.threshold.2 = 0.2, cf.nperm.2 = 1, predList = NULL, folder = NULL,
PLOTfn = NULL, device.type = NULL, res=NULL, jpeg.res = 72,
device.width = 7,  device.height = 7, units="in", pointsize=12,
cex=par()$cex,...)
```

Arguments

`model.obj.1`	R model object. The model object to use for left side of barchart. The model object must be of type `"RF"` (random forest), `"QRF"` (quantile random forest), or `"CF"` (conditional forest). The `ModelMap` package does not currently support `SGB` models. The model object must have the exact same predictors as `model.obj.2`.
`model.obj.2`	R model object. The model object to use for right side of barchart. The model object must be of type `"RF"` (random forest), `"QRF"` (quantile random forest), or `"CF"` (conditional forest). The `ModelMap` package does not currently support `SGB` models. The model object must have the exact same predictors as `model.obj.1`.
`model.name.1`	String. Label for left side of barchart.
`model.name.2`	String. Label for right side of barchart.
`imp.type.1`	Number. Type of importance to use for model 1. Importance type 1 is permutation based, as described in Breiman (2001). Importance type 2 is model based. For RF models is the decrease in node impurities attributable to each predictor variable. For SGB models, it is the reduction attributable to each variable in predicting the gradient on each iteration. Default for random forest models is `imp.type.1 = 1`.
`imp.type.2`	Number. Type of importance to use for model 2. Importance type 1 is permutation based, as described in Breiman (2001). Importance type 2 is model based. For RF models is the decrease in node impurities attributable to each predictor variable. For SGB models, it is the reduction attributable to each variable in

predicting the gradient on each iteration. Default for random forest models is `imp.type.2 = 1`.

`type.label` Logical. Should axis labels include importance type for each side of plot.

`class.1` String. For binary and categorical random forest models. If the name a class is specified, the class-specific relative influence is used for plot. If `class.1 = NULL` overall relative influence used for plot.

`class.2` String. For binary and categorical random forest models. If the name a class is specified, the class-specific relative influence is used for plot. If `class.2 = NULL` overall relative influence used for plot.

`quantile.1` Numeric. QRF models. Quantile to use for model 1. Must be one of the quantiles used in building the QRF model.

`quantile.2` Numeric. QRF models. Quantile to use for model 2. Must be one of the quantiles used in building the QRF model.

`col.1` String. For binary and categorical random forest models. Color to use for bars for model 1. Defaults to grey.

`col.2` String. For binary and categorical random forest models. Color to use for bars for model 2. Defaults to black.

`scale.by` String. Scale by: `"max"` or `"sum"`. When `scale.by="max"` the importance are scaled for each model so that the maximum importance for each model fills the graph. When `scale.by="sum"`, the importance for each model are scaled to sum to 100.

`sort.by` String. Sort by: `"model.obj.1"`, `"model.obj.2"`, `"predList"`. Gives the order to draw the bars for the predictor variables. When `sort.by="model.obj.1"` the predictors are sorted largest to smallest based on importance from model 1. When `sort.by="model.obj.2"` the predictors are sorted largest to smallest based on importance from model 2. When `sort.by="predList"` the predictors are sorted to match the order given in `"predList"`.

`cf.mincriterion.1`
Number. CF models. The value of the test statistic or 1 - p-value that must be exceeded in order to include a split in the computation of the importance. The default `cf.mincriterion.1 = 0` guarantees that all splits are included.

`cf.conditional.1`
Logical. CF models. A logical determining whether unconditional or conditional computation of the importance is performed for `model.obj.1`.

`cf.threshold.1` Number. CF models. The value of the test statistic or 1 - p-value of the association between the variable of interest and a covariate that must be exceeded inorder to include the covariate in the conditioning scheme for the variable of interest (only relevant if `conditional = TRUE`).

`cf.nperm.1` Number. CF models. The number of permutations performed.

`cf.mincriterion.2`
Number. CF models. The value of the test statistic or 1 - p-value that must be exceeded in order to include a split in the computation of the importance. The default `cf.mincriterion.2 = 0` guarantees that all splits are included.

`cf.conditional.2`
: Logical. CF models. A logical determining whether unconditional or conditional computation of the importance is performed for `model.obj.2`.

`cf.threshold.2`
: Number. CF models. The value of the test statistic or 1 - p-value of the association between the variable of interest and a covariate that must be exceeded inorder to include the covariate in the conditioning scheme for the variable of interest (only relevant if `conditional = TRUE`).

`cf.nperm.2`
: Number. CF models. The number of permutations performed.

`predList`
: String. A character vector of the predictor short names used to build the models. If `sort.by="predList"`, then `predList` is used to specify the order to draw the predictors in the barchart.

`folder`
: String. The folder used for all output. Do not add ending slash to path string. If `folder = NULL` (default), a GUI interface prompts user to browse to a folder. To use the working directory, specify `folder = getwd()`.

`PLOTfn`
: String. The file name to use to save the generated graphical plots. If `PLOTfn = NULL` a default name is generated by pasting `model.name.1_model.name.2`. The filename can be the full path, or it can be the simple basename, in which case the output will be to the folder specified by `folder`.

`device.type`
: String or vector of strings. Model validation. One or more device types for graphical output from model validation diagnostics.

  Current choices:

`"default"`	default graphics device
`"jpeg"`	*.jpg files
`"none"`	no graphics device generated
`"pdf"`	*.pdf files
`"png"`	*.png files
`"postscript"`	*.ps files
`"tiff"`	*.tif files

`res`
: Integer. Model validation. Pixels per inch for jpeg, png, and tiff plots. The default is 72dpi, good for on screen viewing. For printing, suggested setting is 300dpi.

`jpeg.res`
: Integer. Model validation. Deprecated. Ignored unless `res` not provided.

`device.width`
: Integer. Model validation. The device width for diagnostic plots in inches.

`device.height`
: Integer. Model validation. The device height for diagnostic plots in inches.

`units`
: Model validation. The units in which `device.height` and `device.width` are given. Can be `"px"` (pixels), `"in"` (inches, the default), `"cm"` or `"mm"`.

`pointsize`
: Integer. Model validation. The default pointsize of plotted text, interpreted as big points (1/72 inch) at `res` ppi

`cex`
: Integer. Model validation. The cex for diagnostic plots.

`...`
: Arguments to be passed to methods, such as graphical parameters (see `par`).

Details

The importance measures used in this plot depend on the model type (RF verses SGB) and the response type (continuous, categorical, or binary).

Importance type 1 is permutation based, as described in Breiman (2001). Importance is calculated by randomly permuting each predictor variable and computing the associated reduction in predictive performance using Out Of Bag error for RF models and training error for SGB models. Note that for SGB models permutation based importance measures are still considered experimental. Importance type 2 is model based. For RF models, importance type 2 is calculated by the decrease in node impurities attributable to each predictor variable. For SGB models, importance type 2 is the reduction attributable to each variable in predicting the gradient on each iteration as described in described in Friedman (2001).

For RF models:

response type	type		Importance Measure
"continuous"	1	permutation	%IncMSE
"binary"	1	permutation	Mean Decrease Accuracy
"categorical"	1	permutation	Mean Decrease Accuracy
"continuous"	2	node impurity	Residual sum of squares
"binary"	2	node impurity	Mean Decrease Gini
"categorical"	2	node impurity	Mean Decrease Gini

For Random Forest models, if `imp.type` not specified, importance type defaults to `imp.type` of `1` - permutation importance. For SGB models, permutation importance is considered experimental so importance defaults to `imp.type` of `2` - reduction of gradient of the loss function.

Also, for binary and categorical Random Forest models, class specific importance plots can be generated by the use of the `class` argument. Note that class specific importance is only available for Random Forest models with importance type 1.

For CF models:

response type	type		Importance Measure
"continuous"	1	permutation	Mean Decrease Accuracy
"binary"	1	permutation	Mean Decrease Accuracy
"categorical"	1	permutation	Mean Decrease Accuracy
"continuous"	2	node impurity	Not Available
"binary"	2	node impurity	Mean Decrease in AUC
"categorical"	2	node impurity	Not Available

For binary CF models, ifimportance.type = 2, function uses AUC-based variables importances as described by Janitza et al. (2012). Here, the area under the curve instead of the accuracy is used to calculate the importance of each variable. This AUC-based variable importance measure is more robust towards class imbalance.

Also, for CF models, if `cf.conditional = TRUE`, the importance of each variable is computed by permuting within a grid defined by the covariates that are associated (with 1 - p-value greater than threshold) to the variable of interest. The resulting variable importance score is conditional in the sense of beta coefficients in regression models, but represents the effect of a variable in both main effects and interactions. See Strobl et al. (2008) for details. Conditional improtance can be slow for

large datasets.

Value

The function returns a two element list: `IMP1` is the variable importance for `model.obj.1`; and, `IMP2` is the variable importance for `model.obj.2`. This is mostly intended for CF models, where calculating the conditional importance can represent a considerable time investment. For other model types it would be just as easy to recalcuate importances on the fly as needed.

Note

Importance currently unavailable for QRF models.

Author(s)

Elizabeth Freeman

References

Breiman, L. (2001) Random Forests. Machine Learning, 45:5-32.

Alexander Hapfelmeier, Torsten Hothorn, Kurt Ulm, and Carolin Strobl (2012). A New Variable Importance Measure for Random Forests with Missing Data. Statistics and Computing, http://dx.doi.org/10.1007/s11222-012-9349-1

Torsten Hothorn, Kurt Hornik, and Achim Zeileis (2006b). Unbiased Recursive Partitioning: A Conditional Inference Framework. Journal of Computational and Graphical Statistics, 15 (3), 651-674. Preprint available from http://statmath.wu-wien.ac.at/~zeileis/papers/Hothorn+Hornik+Zeileis-2006.pdf

Silke Janitza, Carolin Strobl and Anne-Laure Boulesteix (2013). An AUC-based Permutation Variable Importance Measure for Random Forests. BMC Bioinformatics.2013, 14 119. http://www.biomedcentral.com/1471-2105/14/119

Carolin Strobl, Anne-Laure Boulesteix, Thomas Kneib, Thomas Augustin, and Achim Zeileis (2008). Conditional Variable Importance for Random Forests. BMC Bioinformatics, 9, 307. http://www.biomedcentral.com/1471-2105/9/307

See Also

`model.build`

Examples

```
## Not run:

##############################################################################
############################## Run this set up code: #########################
##############################################################################

# set seed:
seed=38
```

```
# Define training and test files:

qdata.trainfn = system.file("extdata", "helpexamples","DATATRAIN.csv", package = "ModelMap")

# Define folder for all output:
folder=getwd()


#identifier for individual training and test data points

unique.rowname="ID"

######################################################################
########## Continuous Response, Continuous Predictors ############
######################################################################

#file names:
MODELfn.RF="RF_Bio_TC"

#predictors:
predList=c("TCB","TCG","TCW")

#define which predictors are categorical:
predFactor=FALSE

# Response name and type:
response.name="BIO"
response.type="continuous"

########## Build Models ################################

model.obj.RF = model.build( model.type="RF",
                        qdata.trainfn=qdata.trainfn,
                        folder=folder,
                        unique.rowname=unique.rowname,
                        MODELfn=MODELfn.RF,
                        predList=predList,
                        predFactor=predFactor,
                        response.name=response.name,
                        response.type=response.type,
                        seed=seed
)

########## Make Imortance Plot - RF Importance type 1 vs 2 #######

model.importance.plot( model.obj.1=model.obj.RF,
model.obj.2=model.obj.RF,
model.name.1="PercentIncMSE",
model.name.2="IncNodePurity",
imp.type.1=1,
imp.type.2=2,
scale.by="sum",
```

```
sort.by="predList",
predList=predList,
main="Imp type 1 vs Imp type 2",
device.type="default")


#####################################################################
########## Categorical Response, Continuous Predictors ##########
#####################################################################

file name:
MODELfn="RF_NLCD_TC"

predictors:
predList=c("TCB","TCG","TCW")

define which predictors are categorical:
predFactor=FALSE

 Response name and type:
response.name="NLCD"
response.type="categorical"

########## Build Model ###################################

model.obj.NLCD = model.build( model.type="RF",
                       qdata.trainfn=qdata.trainfn,
                       folder=folder,
                       unique.rowname=unique.rowname,
                       MODELfn=MODELfn,
                       predList=predList,
                       predFactor=predFactor,
                       response.name=response.name,
                       response.type=response.type,
                       seed=seed)

############## Make Imortance Plot ##################

model.importance.plot( model.obj.1=model.obj.NLCD,
model.obj.2=model.obj.NLCD,
model.name.1="NLCD=41",
model.name.2="NLCD=42",
class.1="41",
class.2="42",
scale.by="sum",
sort.by="predList",
predList=predList,
main="Class 41 vs. Class 42",
device.type="default")


#####################################################################
############## Conditonal inference forest models ################
```

```
#####################################################################

#predictors:
predList=c("TCB","TCG","TCW","NLCD")

#define which predictors are categorical:
predFactor=c("NLCD")

#binary response
response.name="CONIFTYP"
response.type="binary"
MODELfn.CF="CF_CONIFTYP_TCandNLCD"

####################### Build Model ##############################

model.obj.CF = model.build( model.type="CF",
                        qdata.trainfn=qdata.trainfn,
                        folder=folder,
                        unique.rowname=unique.rowname,
                        MODELfn=MODELfn.CF,
                        predList=predList,
                        predFactor=predFactor,
                        response.name=response.name,
                        response.type=response.type,
                        seed=seed
)

################## Make Imortance Plot #########################

#Conditional vs. Unconditional importance#

model.importance.plot( model.obj.1=model.obj.CF,
model.obj.2=model.obj.CF,
model.name.1="conditional",
model.name.2="unconditional",
imp.type.1=1,
imp.type.2=1,
cf.conditional.1=TRUE,
cf.conditional.2=FALSE,
scale.by="sum",
sort.by="predList",
predList=predList,
main="Conditional verses Unconditional",
device.type="default"
)


## End(Not run) # end dontrun
```

`model.interaction.plot` *plot of two-way model interactions*

Description

Image or Perspective plot of two-way model interactions. Ranges of two specified predictor variables are plotted on X and Y axis, and fitted model values are plotted on the Z axis. The remaining predictor variables are fixed at their mean (for continuous predictors) or their most common value (for categorical predictors).

Usage

```
model.interaction.plot(model.obj = NULL, x = NULL, y = NULL,
response.category=NULL, quantiles=NULL, all=FALSE, obs=1, qdata.trainfn = NULL,
folder = NULL, MODELfn = NULL,  PLOTfn = NULL, pred.means = NULL,
xlab = NULL, ylab = NULL, x.range = NULL, y.range = NULL,
z.range = NULL, ticktype = "detailed", theta = 55, phi = 40,
smooth = "none", plot.type = NULL, device.type = NULL,
res=NULL, jpeg.res = 72, device.width = 7,  device.height = 7,
units="in", pointsize=12, cex=par()$cex,
col = NULL, xlim = NULL, ylim = NULL, zlim = NULL, ...)
```

Arguments

`model.obj`	R model object. The model object to use for prediction. The model object must be of type `"RF"` (random forest), `"QRF"` (quantile random forest), or `"CF"` (conditional forest). The `ModelMap` package does not currently support `SGB` models.
`x`	String or Integer. Name of predictor variable to be plotted on the x axis. Alternativly, can be a number indicating a variable name from `predList`.
`y`	String or Integer. Name of predictor variable to be plotted on the y axis. Alternatively, can be a number indicating a variable name from `predList`.
`response.category`	String. Used for categorical response models. Specify which category of response variable to use. This category's probabilities will be plotted on the z axis.
`quantiles`	Numeric. Used for QRF models. Specify which quantile of response variable to use. This quantile will be plotted on the z axis. Note: unlike other functions `model.interaction.plot` will only use a single quantile. If `quantiles` is a vector only the first value will be used.
`all`	Logical. Used for QRF models. A logical value. `all=TRUE` uses all observations for prediction. `all=FALSE` uses only a certain number of observations per node for prediction (set with argument `obs`). The default is `all=FALSE`.
`obs`	Numeric. Used for QRF models. An integer number. Determines the maximal number of observations per node to use for prediction. The input is ignored for all=TRUE. The default is obs=1.

qdata.trainfn — String. The name (full path or base name with path specified by folder) of the training data file used for building the model (file should include columns for both response and predictor variables). The file must be a comma-delimited file *.csv with column headings. qdata.trainfn can also be an R dataframe. If predictions will be made (predict = TRUE or map=TRUE) the predictor column headers must match the names of the raster layer files, or a rastLUT must be provided to match predictor columns to the appropriate raster and band. If qdata.trainfn = NULL (the default), a GUI interface prompts user to browse to the training data file.

folder — String. The folder used for all output. Do not add ending slash to path string. If folder = NULL (default), a GUI interface prompts user to browse to a folder. To use the working directory, specify folder = getwd().

MODELfn — String. The file name used to save the generated model object, only used if PLOTfn = NULL. If MODELfn is supplied and If PLOTfn = NULL, a graphical file name is generated by pasting MODELfn_plot.type_x.name_y.name. If PLOTfn = NULL and MODELfn = NULL, a default name is generated by pasting model.type_response.type_respo
The filename can be the full path, or it can be the simple basename, in which case the output will be to the folder specified by folder.

PLOTfn — String. The file name to use to save the generated graphical plots. The filename can be the full path, or it can be the simple basename, in which case the output will be to the folder specified by folder.

pred.means — Vector. Allows specification of values for other predictor variables. If Null, other predictors are set to their mean value (for continuous predictors) or their most common value (for factored predictors).

xlab — String. Allows manual specification of the x label.

ylab — String. Allows manual specification of the y label.

x.range — Vector. Manual range specification for the x axis. Alternate argument name for xlim. Use one or the other. Do not provide both x.range and xlim.

y.range — Vector. Manual range specification for the y axis. Alternate argument name for ylim. Use one or the other. Do not provide both y.range and ylim.

z.range — Vector. Manual range specification for the z axis. Alternate argument name for zlim. Use one or the other. Do not provide both z.range and zlim.

ticktype — Character: "simple" draws just an arrow parallel to the axis to indicate direction of increase; "detailed" (default) draws normal ticks as per 2D plots. If X or y is factored, ticks will be drawn on both axes.

theta — Numeric. Angles defining the viewing direction. theta gives the azimuthal direction.

phi — Numeric. Angles defining the viewing direction. phi gives the colatitude.

smooth — String. controls smoothing of the predicted surface. Options are "none" (default), "model" which uses a glm model to smooth the surface, and "average" which applies a 3x3 smoothing average. Note: smoothing is not appropriate if X or y is factored.

plot.type — Character. "persp" gives a 3-D perspective plot. "image" gives an image plot.

`device.type` String or vector of strings. Model validation. One or more device types for graphical output from model validation diagnostics.
Current choices:

"default"	default graphics device
"jpeg"	*.jpg files
"none"	no graphics device generated
"pdf"	*.pdf files
"postscript"	*.ps files
"win.metafile"	*.emf files

res	Integer. Model validation. Pixels per inch for jpeg, png, and tiff plots. The default is 72dpi, good for on screen viewing. For printing, suggested setting is 300dpi.
jpeg.res	Integer. Model validation. Deprecated. Ignored unless res not provided.
device.width	Integer. Model validation. The device width for diagnostic plots in inches.
device.height	Integer. Model validation. The device height for diagnostic plots in inches.
units	Model validation. The units in which device.height and device.width are given. Can be "px" (pixels), "in" (inches, the default), "cm" or "mm".
pointsize	Integer. Model validation. The default pointsize of plotted text, interpreted as big points (1/72 inch) at res ppi
cex	Integer. Model validation. The cex for diagnostic plots.
col	Vector. Color table to use for image plots (see help file on image for details).
xlim	Vector. X limits. Alternate argument name for x.range. Use one or the other. Do not provide both x.range and xlim.
ylim	Vector. Y limits. Alternate argument name for y.range. Use one or the other. Do not provide both y.range and ylim.
zlim	Vector. Z limits. Alternate argument name for z.range. Use one or the other. Do not provide both z.range and zlim.
...	additional graphical parameters (see par).

Details

This function provides a diagnostic plot useful in visualizing two-way interactions between predictor variables. Two of the predictor variables from the model are used to produce a grid of possible combinations of predictor values over the range of both variables. The remaining predictor variables from the model are fixed at either their means (for continuous predictors) or their most common value (for categorical predictors). Model predictions are generated over this grid and plotted as the z axis.

This function works with both continuous and categorical predictors, though the perspective plot should be interpreted with care for categorical predictors. In particular, the smooth option is not appropriate if either of the two selected predictor variables is categorical.

For categorical response models, a particular value must be specified for the response using the response.category argument.

Author(s)

Elizabeth Freeman

References

This function is adapted from gbm.perspec version 2.9 April 2007, J Leathwick/J Elith. See appendix S3 from:

Elith, J., Leathwick, J. R. and Hastie, T. (2008). A working guide to boosted regression trees. Journal of Animal Ecology. 77:802-813.

Examples

```
## Not run:

#############################################################################
############################## Run this set up code: ########################
#############################################################################

# set seed:
seed=38

# Define training and test files:

qdata.trainfn = system.file("extdata", "helpexamples","DATATRAIN.csv", package = "ModelMap")
qdata.testfn = system.file("extdata", "helpexamples","DATATEST.csv", package = "ModelMap")

# Define folder for all output:
folder=getwd()

########## Continuous Response, Categorical Predictors ############


#file name to store model:
MODELfn="RF_BIO_TCandNLCD"

#predictors:
predList=c("TCB","TCG","TCW","NLCD")

#define which predictors are categorical:
predFactor=c("NLCD")

# Response name and type:
response.name="BIO"
response.type="continuous"

#identifier for individual training and test data points

unique.rowname="ID"


#############################################################################
############################## build model: #################################
#############################################################################
```

```
### create model ###

model.obj = model.build( model.type="RF",
                        qdata.trainfn=qdata.trainfn,
                        folder=folder,
                        unique.rowname=unique.rowname,
                        MODELfn=MODELfn,
                        predList=predList,
                        predFactor=predFactor,
                        response.name=response.name,
                        response.type=response.type,
                        seed=seed,
                        na.action=na.roughfix
)

#############################################################################
###################### make interaction plots: ##############################
#############################################################################

#########################
### Perspective Plots ###
#########################


### specify first and third  predictors in 'predList (both continuous) ###

model.interaction.plot( model.obj,
x=1,y=3,
main=response.name,
plot.type="persp",
device.type="default")


### specify predictors in 'predList' by name (one continuous one factored) ###

model.interaction.plot(model.obj,
x="TCB", y="NLCD",
main=response.name,
plot.type="persp",
device.type="default")


###################
### Image Plots ###
###################

### same as previous example, but image plot ###


l <- seq(100,0,length.out=101)
c <- seq(0,100,length.out=101)
col.ramp <- hcl(h = 120, c = c, l = l)
```

```
model.interaction.plot( model.obj,
x="TCB", y="NLCD",
main=response.name,
plot.type="image",
device.type="default",
col = col.ramp)


#########################
### 3-way Interaction ###
#########################

### use 'pred.means' argument to fix values of additional predictors ###

### factored 3rd predictor ###

interaction between TCG and TCW for 3 most common values of NLCD

nlcd<-levels(model.obj$predictor.data$NLCD)
nlcd.counts<-table(model.obj$predictor.data$NLCD)
nlcd.ordered<-nlcd[order(nlcd.counts,decreasing=TRUE)]

for(i in nlcd.ordered[1:3]){
pred.means=list(NLCD=i)

model.interaction.plot( model.obj,
x="TCG", y="TCW",
main=paste("NLCD=",i," (",nlcd.counts[i]," plots)", sep=""),
pred.means=pred.means,
z.range=c(0,110),
theta=290,
plot.type="persp",
device.type="default")
}



### continuos 3rd predictor ###


tcb<-seq( min(model.obj$predictor.data$TCB),
max(model.obj$predictor.data$TCB),
length=3)

tcb<-signif(tcb,2)

for(i in tcb){
pred.means=list(TCB=i)

model.interaction.plot( model.obj,
x="TCG", y="TCW",
```

```
main=paste("TCB =",i),
pred.means=pred.means,
z.range=c(0,120),
theta=290,
plot.type="persp",
device.type="default")
}



### 4-way Interesting combos ###


tcb=c(1300,2900,3400)
nlcd=c(11,90,95)

for(i in 1:3){
pred.means=list(TCB=tcb[i],NLCD=nlcd[i])

model.interaction.plot( model.obj,
x="TCG", y="TCW",
main=paste("TCB =",tcb[i],"        NLCD =",nlcd[i]),
pred.means=pred.means,
z.range=c(0,120),
theta=290,
plot.type="persp",
device.type="default")
}

## End(Not run) #end dontrun
```

`model.mapmake` *Map Making*

Description

Applies models to either ERDAS Imagine image (.img) files or ESRI Grids of predictors to create detailed prediction surfaces. It will handle large predictor files for map making, by reading in the `.img` files in rows, and output to the `.img` file the prediction for each data row, before reading the next row of data.

Usage

```
model.mapmake(model.obj= NULL, folder = NULL, MODELfn = NULL,
rastLUTfn = NULL, na.action = NULL, na.value=-9999, keep.predictor.brick=FALSE,
map.sd = FALSE, OUTPUTfn = NULL, quantiles=NULL)
```

Arguments

model.obj	R model object. The model object to use for prediction. The model object must be of type "RF" (random forest), "QRF" (quantile random forest), or "CF" (conditional forest). The ModelMap package does not currently support SGB models.
folder	String. The folder used for all output from predictions and/or maps. Do not add ending slash to path string. If folder = NULL (default), a GUI interface prompts user to browse to a folder. To use the working directory, specify folder = getwd().
MODELfn	String. The file name to use to save the generated model object. If MODELfn = NULL (the default), a default name is generated by pasting model.type_response.type_response.name. If the other output filenames are left unspecified, MODELfn will be used as the basic name to generate other output filenames. The filename can be the full path, or it can be the simple basename, in which case the output will be to the folder specified by folder.

rastLUTfn	String. The file name (full path or base name with path specified by folder) of a .csv file for a rastLUT. Alternatively, a dataframe containing the same information. The rastLUT must include 3 columns: (1) the full path and name of the raster file; (2) the shortname of each predictor / raster layer (band); (3) the layer (band) number. The shortname (column 2) must match the names predList, the predictor column names in training/test data set (qdata.trainfn and qdata.testfn, and the predictor names in model.obj.
	Example of comma-delimited file:
	`C:/button_test/tc99_2727subset.img, tc99_2727subsetb1, 1`
	`C:/button_test/tc99_2727subset.img, tc99_2727subsetb2, 2`
	`C:/button_test/tc99_2727subset.img, tc99_2727subsetb3, 3`

na.action	String. Model validation. Specifies the action to take if there are NA values in the prediction data or if there is a level or class of a categorical predictor variable in the validation test set or the production (mapping) data set, but not in the training data set. Currently, the only supported option for map making is na.action = "na.omit" (the default) where any data point or pixel with any new levels for any of the factored predictors is returned as NA (na.value, defaults to -9999).
na.value	Number. Value that indicates NA in the predictor rasters. Defaults to -9999. Note: all predictor rasters must use the same value for NA.
keep.predictor.brick	Logical. Map Production. If TRUE then the raster brick containing the predictors from the model object is saved as a native raster package format file. If FALSE a temporary brick is created and then deleted at the end of map production.
map.sd	Logical. Map Production. If map.sd = TRUE, maps of mean, standard deviation, and coefficient of variation of the predictions from all the trees are generated for each pixel. If map.sd = FALSE (the default), only the predicted probability map will be built. This option is only available if the model.type = "RF" and the response.type = "continuous". Note: This option requires much more available memory. The names of the additional maps default to:

```
folder/model.type_response.type_response.name_mean.img
folder/model.type_response.type_response.name_stdev.img
folder/model.type_response.type_response.name_coefv.img
```

OUTPUTfn
: String. Map Production. Filename of output file for map production. The filename can be the full path, or it can be the simple basename, in which case the output will be to the folder specified by `folder`. If `OUTPUTfn = NULL` (the default), a name is created by pasting `modelfn` and `"_map"`.

  The `raster` package uses the filename extension to determine the output format of the map object. See the `raster` package function `writeFormats` for a list of valid write formats. Because of this, only use the `"."` in output filenames if it is indicating a valid file extension.

  If the output filename does not include an extension, the default extension of `".img"` will be added.

  For continuous random forest models with `map.sd = TRUE` then `OUTPUTfn` is also used to create output file names for maps of the mean, standard deviation and coeficient of variation of all the trees predictions for each pixel.

quantiles
: Numeric Vector. QRF models. The quantiles to predict. A numeric vector with values between zero and one. The quantile map output will be a multilayer raster with one layer for each quantile. If `model.obj` was created with the `ModelMap` package, there will also be a single layer outputraster giving the ordinary RF mean predicted values.

Details

`model.mapmake()` can be run in a traditional R command mode, where all arguments are specified in the function call. However it can also be used in a full push button mode, where you type in the simple command `model.mapmake()`, and GUI pop up windows will ask questions about the type of model, the file locations of the data, etc...

When running `model.mapmake()` on non-Windows platforms, file names and folders need to be specified in the argument list, but other pushbutton selections are handled by the `select.list()` function, which is platform independent.

The R package `raster` is used to read spatial rasters into R. The data for production mapping should be in the form of pixel-based raster layers representing the predictors in the model. If there is more than one predictor or raster layer, the layers must all have the same number of columns and rows. The layers must also have the same extent, projection, and pixel size, for effective model development and accuracy. The `raster` package function `compareRaster()` is used to check predictor layers for consistency.

The layers must also be in (single or multi-band) raster data formats that can be read by package `raster`, for example ESRI Grid or ERDAS Imagine image files. The predictor layers must have continuous or categorical data values. See `writeRaster` for a list of available formats.

To improve processing speed, the `raster` package is used to create a raster brick object with a layer for each predictor in the model. By default, this brick is a temporary file that is automatically deleated as soon as the map is completed. If `keep.predictor.brick=TRUE`, the predictor brick with be saved as a native `raster` package file, with a file name created by appending `'_brick'` to the `OUTPUTfn`. Warning: these bricks can be quite large, as they contain all the predictor data for every pixel in the map.

When creating maps of non-rectangular study regions there may be large portions of the rectangle where you have no predictors, and are uninterested in making predictions. The suggested value for the pixels outside the study area is `-9999`. These pixels will be ignored in the predictions, thus saving computing time.

The function `model.mapmake()` outputs an rater file of map information suitable to be imported into a GIS. Maps can also be imported back into R using the function `raster()` from the `raster` package. The file extension of `OUTPUTfn` determines the write format. If `OUTPUTfn` does not include a file extension, output will default to an ERDAS Imagine image file with extension `".img"`

For Binary response models the output is in the form of predicted probability of presence for each pixel. For Continuous response models the output is the predicted value for each pixel. For Categorical response models the map output depends on the category labels. If the categorical response variable is numeric, the map output will use the original numeric categories. If the categories are non-numeric (for example, character strings), map output is in the form of integer class codes for each pixel, coded for each level of the factored response, and a CSV file containing a look up table is also generated to associate the integer codes with the original values of the response categories.

The first predictor from `predList` is used to determine projection of output Imagine Image file.

Value

The `model.mapmake()` function does not return a value, instead it writes a raster file of map information (suitable for importing into a GIS) to the specified folder. The output raster is saved in the format specifed by the file extension of `OUTPUTfn`

The `model.mapmake()` function also writes a text file listing the projections of all predictor rasters.

For categorical response models, a csv file map key is written giving the integer code associated with each response category.

If `keep.predictor.brick = TRUE` then a raster brick of all the predictor rasters from the model is also saved to the specified folder. If `keep.predictor.brick = FALSE` (the default) then the predictor brick is written to a temprary file, and deleted. Warning: the predictor bricks can be quite large, and saving them can require quite a bit of memory.

Note

If `model.mapmake()` is interupted it may leave orphan `.gri` and `.grd` files in your temporary directory. The `raster` package functions `showTmpFiles` and `removeTmpFiles` can be used to locate and remove these files, or they can be deleated manually from the temporary directory.

Author(s)

Elizabeth Freeman and Tracey Frescino

References

Breiman, L. (2001) Random Forests. Machine Learning, 45:5-32.

Liaw, A. and Wiener, M. (2002). Classification and Regression by randomForest. R News 2(3), 18–22.

Ridgeway, G., (1999). The state of boosting. Comp. Sci. Stat. 31:172-181

Simpson, E. H. (1949). Measurement of diversity. Nature.

See Also

get.test, model.build, model.diagnostics, compareRaster, writeRaster

Examples

```
## Not run:

################################################################################
############################## Run this set up code: ###########################
################################################################################

# set seed:
seed=38

# Define training and test files:

qdata.trainfn = system.file("extdata", "helpexamples","DATATRAIN.csv", package = "ModelMap")

# Define folder for all output:
folder=getwd()


#identifier for individual training and test data points

unique.rowname="ID"


################################################################################
######################### Define the model: ####################################
################################################################################


########## Continuous Response, Continuous Predictors ############

#file name to store model:
MODELfn="RF_Bio_TC"

#predictors:
predList=c("TCB","TCG","TCW")

#define which predictors are categorical:
predFactor=FALSE

# Response name and type:
response.name="BIO"
response.type="continuous"



################################################################################
############################## build model: ####################################
```

```
##############################################################################


### create model ###

model.obj = model.build( model.type="RF",
                        qdata.trainfn=qdata.trainfn,
                        folder=folder,
                        unique.rowname=unique.rowname,
                        MODELfn=MODELfn,
                        predList=predList,
                        predFactor=predFactor,
                        response.name=response.name,
                        response.type=response.type,
                        seed=seed,
                        na.action="na.roughfix"
)



##############################################################################
############ Then Run this code to predict map pixels ########################
##############################################################################


### Create a the filename (including path) for the rast Look up Tables ###


rastLUTfn.2001 <- system.file( "extdata",
"helpexamples",
"LUT_2001.csv",
package="ModelMap")




### Load rast LUT table, and add path to the predictor raster filenames in column 1 ###

rastLUT.2001 <- read.table(rastLUTfn.2001,header=FALSE,sep=",",stringsAsFactors=FALSE)

for(i in 1:nrow(rastLUT.2001)){
rastLUT.2001[i,1] <- system.file("extdata",
"helpexamples",
rastLUT.2001[i,1],
package="ModelMap")
}


### Define filename for map  output ###

OUTPUTfn.2001 <- "RF_BIO_TCandNLCD_01.img"
OUTPUTfn.2001 <- paste(folder,OUTPUTfn.2001,sep="/")
```

```
### Create image files of predicted map data ###

model.mapmake( model.obj=model.obj,
               folder=folder,
               rastLUTfn=rastLUT.2001,
           # Mapping arguments
               OUTPUTfn=OUTPUTfn.2001
               )


##############################################################################
################ run this code to create maps in R ###########################
##############################################################################

### Define Color Ramp ###

l <- seq(100,0,length.out=101)
c <- seq(0,100,length.out=101)
col.ramp <- hcl(h = 120, c = c, l = l)


### read in map data ###

mapgrid.2001 <- raster(OUTPUTfn.2001)

#mapgrid.2001 <- setMinMax(mapgrid.2001)


### create map ###

dev.new(width = 5, height = 5)
opar <- par(mar=c(3,3,2,1),oma=c(0,0,3,4),xpd=NA)

zlim <- c(0,max(maxValue(mapgrid.2001)))
legend.label<-rev(pretty(zlim,n=5))
legend.colors<-col.ramp[trunc((legend.label/max(legend.label))*100)+1]

image(  mapgrid.2001, col = col.ramp, zlim=zlim, asp=1, bty="n",
        xaxt="n", yaxt="n", main="", xlab="", ylab="")
mtext("2001 Imagery",side=3,line=1,cex=1.2)

legend( x=xmax(mapgrid.2001),y=ymax(mapgrid.2001),
legend=legend.label,
fill=legend.colors,
bty="n",
cex=1.2
)
mtext("Predictions",side=3,line=1,cex=1.5,outer=TRUE)
par(opar)
```

```
## End(Not run) # end dontrun
```

Index