Package ‘TDMR’

February 19, 2015

Type Package
Title Tuned Data Mining in R
Version 1.2
Date 30.08.2014
Author Wolfgang Konen <wolfgang.konen@fh-koeln.de>, Patrick Koch
  <patrick.koch@fh-koeln.de>
Maintainer Wolfgang Konen <wolfgang.konen@fh-koeln.de>
Description Tuned Data Mining in R (TDMR) performs a complete tuning of a data
  mining task.
License GPL (>= 2)
Depends R (>= 2.14.0), testit, SPOT
Suggests adabag, cmaes, parallel, e1071, powell, ROCR, randomForest,
  rCMA, rSFA, twiddler
Collate ‘makeTdmRandomSeed.r’ ‘printTDMclassifier.r’
  ‘printTDMregressor.r’ ‘tdmBigLoop.r’ ‘tdmClassify.r’
  ‘tdmClassifyLoop.r’ ‘tdmDefaultsFill.r’ ‘tdmDispatchTuner.r’
  ‘tdmEmbedDataFrame.r’ ‘tdmEnvTMakeNew.r’ ‘tdmGeneralUtils.r’
  ‘tdmGetObj.r’ ‘tdmGraphicUtils.r’ ‘tdmMapDesign.r’
  ‘tdmMetacostRf.r’ ‘tdmModelingUtils.r’ ‘tdmOptsDefaults.r’
  ‘tdmParaBootstrap.r’ ‘tdmPlotResMeta.r’ ‘tdmPreprocUtils.r’
  ‘tdmReadData.r’ ‘tdmROCR.r’ ‘tdmRegress.r’ ‘tdmRegressLoop.r’
  ‘tdmSplitTestData.r’ ‘tdmStartSpot.r’ ‘tdmStartOther.r’
  ‘unbiasedBestRun_O.r’ ‘unbiasedRun.r’ ‘tdmExecSpotStep.r’

NeedsCompilation no
Repository CRAN
Date/Publication 2014-08-30 16:14:17

R topics documented:

  TDMR-package .......................................................... 3
  dsetTest.TDMdata .................................................... 4
R topics documented:

dsetTrnVa.TDMdata .............................................. 4
Opts .......................................................... 5
predict.TDMenvir ............................................. 6
print.TDMclassifier ........................................... 7
print.TDMdata .................................................. 8
print.TDMregressor ........................................... 8
tdmBigLoop ..................................................... 9
tdmBindResponse ............................................. 13
tdmClassify .................................................... 13
tdmClassifyLoop ............................................. 16
tdmClassifySummary .......................................... 18
tdmDefaultsFill .............................................. 19
tdmEmbedDataFrame ......................................... 21
tdmEnvTAddBstRes ........................................... 21
tdmEnvTAddGetters ......................................... 22
tdmEnvTLoad .................................................. 22
tdmEnvTMakeNew ............................................ 23
tdmEnvTReadApd ............................................ 24
tdmEnvTSensi ............................................... 24
tdmExecSpotStep ............................................. 25
tdmGraAndLogFinalize ....................................... 26
tdmGraAndLogInitialize .................................... 27
tdmGraphicCloseDev ......................................... 27
tdmGraphicCloseWin ......................................... 28
tdmGraphicInit .............................................. 28
tdmGraphicNewWin .......................................... 29
tdmGraphicToTop ............................................ 29
tdmMapDesApply ............................................. 30
tdmMapDesLoad .............................................. 30
tdmModConfmat ............................................... 31
tdmModCreateCVindex ....................................... 32
tdmModSortedRFimport ..................................... 34
tdmModVote2Target ......................................... 35
tdmOptsDefaultsSet ......................................... 36
tdmParaBootstrap ........................................... 41
tdmPlotResMeta ............................................. 42
tdmPreAddMonomials ....................................... 43
tdmPreFindConstVar ....................................... 44
tdmPreGroupLevels ......................................... 45
tdmPreLevel2Target ........................................ 45
tdmPreNAroughfix .......................................... 46
tdmPrePCA.apply ............................................ 47
tdmPrePCA.train ............................................ 48
tdmPreSFA.apply ............................................ 49
tdmPreSFA.train ............................................ 50
tdmRandomSeed .............................................. 51
tdmReadCmd .................................................. 52
tdmReadData ................................................. 53
TDMR-package

TMDR is a package for tuned data mining (predictive analytics, i.e. classification and regression). Its main features are:

1) A variety of tuners, with special emphasis on SPOT (a well-known R package for parameter tuning), but also CMA-ES (package rCMA) and other tuning algorithms.
2) Tuning of preprocessing (feature generation) parameters and model building parameters simultaneously.
3) Support for multiple tuning experiments (different settings, repetitions with different resamplings, ...).
4) Easy parallelization of those experiments with the help of R packages snow and parallel.
5) Extensibility: New tuning parameters, new feature preprocessing tools, model builders and even new tuners can be added easily.

The main entry point functions are tdmClassifyLoop, tdmRegressLoop and tdmBigLoop. See tdmOptsDefaultsSet and tdmDefaultsFill for an overview of adjustable TDMR-parameters.

Author(s)

Wolfgang Konen (<wolfgang.konen@fh-koeln.de>), Patrick Koch
dsetTest.TDMdata

Return the test part of a TDMdata object containing the task data.

Description
Return the test part of a TDMdata object containing the task data.

Usage
### S3 method for class 'TDMdata'

dsetTest(x)

Arguments

x
return value from a prior call to tdmSplitTestData, an object of class TDMdata.

Value
tset, a data frame with all test records. If there are 0 test records, return NULL.

Author(s)
Wolfgang Konen, FHK, Feb'2012

See Also

unbiasedRun dsetTrnVa.TDMdata

---

dsetTrnVa.TDMdata

Return the train-validation part of a TDMdata object containing the task data.

Description
Return the train-validation part of a TDMdata object containing the task data.

Usage
### S3 method for class 'TDMdata'

dsetTrnVa(x)

---

References

http://gociop.de/research-projects/tuned-data-mining/
**Approaches**

**Arguments**

- `x`: return value from a prior call to `tdmSplitTestData`, an object of class `TDMdata`.

**Value**

- `dset`, a data frame with all train-validation records

**Author(s)**

- Wolfgang Konen, FHK, Feb’2012

**See Also**

- `dsetTest.TDMdata` `tdmSplitTestData`

---

**Description**

Returns the list opts from objects of class `TDMenvir`, `TDMclassifier`, `TDMregressor`, `tdmClass` or `tdmRegre`.

**Usage**

```r
Opts(x, ...)
```

- ## S3 method for class `TDMenvir`
  `Opts(x, ...)`

- ## S3 method for class `TDMclassifier`
  `Opts(x, ...)`

- ## S3 method for class `TDMregressor`
  `Opts(x, ...)`

- ## S3 method for class `tdmClass`
  `Opts(x, ...)`

- ## S3 method for class `tdmRegre`
  `Opts(x, ...)`

- ## Default S3 method:
  `Opts(x, ...)"
** predict.TDMenvir **

** Arguments **

- `x` an object of class `TDClassifier, tdmClass, TDMregressor` or `tdmRegre`.  
- `...` – currently not used –

** Value **

the list `opts` with DM-specific settings contained in the specified object

---

** predict.TDMenvir **

*Make a prediction using the last model.*

** Description **

Make a prediction with objects of class `TDClassifier, TDMclassifier, TDMregressor`. The prediction is based on the (last) model trained during `unbiasedRun`.

** Usage **

```r
## S3 method for class 'TDClassifier'
predict(object, ...)

## S3 method for class 'TDClassifier'
predict(object, ...)

## S3 method for class 'TDClassifier'
predict(object, ...)
```

** Arguments **

- `object` an object of class `TDClassifier, TDMclassifier, TDMregressor` containing in element `lastModel` the relevant model.  
- `...` arguments passed on to the model’s predict function. Usually the first argument of `...` should be `newdata`, a data frame for which new predictions are desired.

** Value **

a vector with length `nrow(newdata)` containing the new predictions.

** Examples **

```r
## Not run:

## This example requires that demo04cpu.r is executed first (it will write demo04cpu.RData)
path <- paste(find.package("TDClassifier"), "demo04cpu", sep="/`/`");  
tdm <- list( filenameEnvT="demo04cpu.RData" );  # file with environment envT
load(paste(path,tdm$filenameEnvT,sep="/`/`");

# take only the first 15 records:
```
print.TDMclassifier

newdata=read.csv2(file=paste(path,"data/cpu.csv", sep=""), dec="."")[1:15,];
z=predict(envT,newdata);
print(z);

## End(Not run)

print.TDMclassifier Print an overview for a TDMclassifier or tdmClass object.

Description

Print an overview for a TDMclassifier or tdmClass object.

Usage

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

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

Arguments

x an object of class tdmClass, as returned from a prior call to tdmClassify,
or an object of class TDMclassifier, as returned from a prior call to tdmClassifyLoop.

... e.g. 'type' which information to print:
"overview" (default) relative gain on training/test set, number of records, see
tdmClassifySummary
"cm.train" confusion matrix on train set
"cm.vali" confusion matrix on test set
"?" help on this method

Author(s)

Wolfgang Konen, FHK, 2011 - 2013

See Also

tdmClassify, tdmClassifySummary, TDMclassifier
print.TDMdata

Print an overview for a TDMdata object.

Description
Print number of rows and number of columns of the data frame dset contained in the TDMdata object.

Usage
```R
## S3 method for class 'TDMdata'
print(x, ...)
```

Arguments
- `x` return value from a prior call to `tdmSplitTestData`, an object of class `TDMdata`.
- `...` currently not used

Author(s)
Wolfgang Konen, FHK

See Also
tdmSplitTestData

print.TDMregressor

Print an overview for a TDMregressor or tdmRegre object.

Description
Print an overview for a TDMregressor or tdmRegre object.

Usage
```R
## S3 method for class 'TDMregressor'
print(x, ...)
```

```R
## S3 method for class 'tdmRegre'
print(x, ...)
```
Arguments

x

- an object of class `tdmRegre`, as returned from a prior call to `tdmRegress`, or an object of class `TDmRegressor`, as returned from a prior call to `tdmRegressLoop`.

... e.g. 'type' which information to print:

- "overview" (def.) RMAE on training/test set, number of records, see `tdmRegressSummary`
- "..." ... other choices, TODO ...
- "?" help on this method

See Also

`tdmRegress`, `tdmRegressSummary`, `TDmRegressor`

dmbigloop

Tuning and unbiased evaluation in a big loop.

Description

For each `.conf` file in `tdm$runList` call all tuning algorithms (SPOT, CMA-ES or other) specified in `tdm$tuneMethod` (via function `tdmDispatchTuner`). After each tuning process perform a run of `tdm$unbiasedFunc` (usually `unbiasedRun`). Each of these experiments is repeated `tdm$nEperim` times. Thus we have for each tripel

```
(confFile,nExp,theTuner)
```

a tuning result. The ranges of the triple elements are:

```
confFile in tdm$runList
nExp in 1,...,tdm$nExperim
theTuner in tdm$tuneMethod
```

Usage

dmbigloop(envT, spotStep = "auto", dataObj = NULL)

Arguments

envT

- an environment containing on input at least the element `tdm` (a list with general settings for TDMR, see `tdmDefaultsFill`), which has at least the elements

  - `tdm$runList` vector of `.conf` filenames
  - `tdm$spotList` [NULL] vector of `.conf` filenames for which spot tuning is done. If NULL, then `spotList=runList`
spotStep: ["auto"] which step of SPOT to execute (either "auto" or "rep").

dataObj: [NULL] optional object of class TDMdata (the same for all runs in big loop). If it is NULL, it will be constructed here with the help of tdmSplitTestData. Then it may be different for each run in the big loop.

details:

tdm refers to envT$tdm.

Tuning is skipped if the .conf file does not appear in tdm$spotList or if spotStep=="rep". In this case it is assumed then that envT$bstGrid and envT$resGrid contain the appropriate data frames already. See tdmEnvTAddBstRes on how to fill envT$bstGrid and envT$resGrid from an .RData file.

The available tuning algorithms (tuners) are

- **spotTuner**: Call spot.
- **lhdtuner**: Perform a parameter tuning using a Latin hypercube design (LHD) for obtaining best design points. LHD is performed by configuring SPOT in such a way that all the budget is used for the initial design (usually LHD).
- **cma_jTuner**: Perform a parameter tuning by CMA-ES, using the *Java* implementation by Niko Hansen through the interface package rCMA.
- **cmaesTuner**: Perform a parameter tuning by CMA-ES, using the *R* implementation (package cmaes by Olaf Mersmann) (deprecated, use cma_jTuner instead).
- **bfgstuner**: Perform a parameter tuning by Broyden, Fletcher, Goldfarb and Shanno (BFGS) method. The L-BFGS-B version allowing box constraints is used.
- **powelltuner**: Perform a parameter tuning by Powell’s UObyQA algorithm (unconstrained optimization by quadratic approximation), see package powell.

value:

environment envT, containing the results

- **res**: data frame with results from last tuning (one line for each call of tdmStart*)
- **bst**: data frame with the best-so-far results from last tuning (one line collected after each (SPO) step)
- **resGrid**: list with data frames res from all tuning runs. Use envT$getRes(envT,confFile,nExp,thetuner) to retrieve a specific res.
- **bstGrid**: list with data frames bst from all tuning runs. Use envT$getBst(envT,confFile,nExp,thetuner) to retrieve a specific bst.
- **theFinals**: data frame with one line for each triple (confFile,nExp,thetuner), each line contains summary information about the tuning run in the form:

```
confFile tuner nExp [params] NRUN NEVAL RGain bst RGain.* sdR.*
```

where [params] is written depending on tdm$withParams.

NRUN is the number of unbiased evaluation runs.
NEVAL is the number of function evaluations (model builds) during tuning. RGain denotes the relative gain on a certain data set: the actual gain achieved with the model divided by the maximum gain possible for the current cost matrix and the current data set. This is for classification tasks, in the case of regression each RGain.* is replaced by RMAE.*, the relative mean absolute error. Each 'sdR.' denotes the standard deviation of the preceeding RGain or RMAE. RGain.bst is the best result during tuning obtained on the training-validation data. RGain.avg is the average result during tuning. The following pairs RGain.* sdR.* are the results of one or several unbiased evaluations on the test data where '*' takes as many values as there are elements in tdm$mode (the possible values are explained in unbiasedRun).

result object of class TDMclassifier or TDMregressor. This is a list with results from tdm$mainFunc as called in the last unbiased evaluation using the best parameters found during tuning. Use print(envT$result) to get more info on such an object of class TDMclassifier.

tunerVal an object with the return value from the last tuning process. For every tuner, this is the list spotConfig, containing the SPOT settings plus the TDMR settings in elements opts and tdm. Every tuner extends this list by tunerVal$alg.currentResult and tunerVal$alg.currentBest, see tmdDispatchTuner. In addition, each tuning method might add specific elements to the list, see the description of each tuner.

Environment envT contains further elements, but they are only relevant for the internal operation of tdmBigLoop and its subfunctions.

Note

Side effects: Irrespective of the value of tdm$fileMode,

- a compressed version of envT is saved to file tdm$filenameEnvT (default: <runList[1]> .RData), relative to the directory of the .conf file.

If tdm$saveModel==TRUE, then envT$result$lastRes$lastModel (the last trained model) will be saved to tdm$filenameEnvT. The default is tdm$saveModel==FALSE (smaller .RData files).

If tdm$fileMode==TRUE, more files are written relative to the directory of the .conf file:

- envT$theFinals is written to file tdm$finalFile
- envT$theFinals is appended to tdm$experFile

If tdm$finalFile==NULL, then it is set to sub(".conf",".fin",runList[1]).
If tdm$experFile==NULL, then nothing is appended to any experiment file.

Example usages of function tdmBigLoop are shown in

demo(demo03sonar)
demo(demo03sonar_B)
demo(demo04cpu)
where the corresponding R-sources are in directory demo.

Author(s)

Wolfgang Konen (<wolfgang.konen@fh-koeln.de>), Patrick Koch

See Also

tdmDispatchTuner, unbiasedRun

Examples

```r
## This demo shows a complete tuned data mining process (level 3 of TDMR) where
## the data mining task is the classification task SONAR (from UCI repository,
## The data mining process is in main_sonar.r, which calls tdmClassifyLoop and tdmClassify
## with Random Forest as the prediction model.
## The three parameter to be tuned are CUTOFF1, CLASSWT2 and XPERC, as specified
## in file sonar_04.roi. The tuner used here is LHD.
## Tuning runs are rather short, to make the example run quickly.
## Do not expect good numeric results.
## See demo/demo03sonar.B.r for a somewhat longer tuning run, with two tuners SPOT and LHD.

## set working directory (dir with .apd, .conf and main_.r file)
path <- paste(find.package("TDMR"), "demo02sonar", sep="/");
source(paste(path,"main_sonar.r",sep="/"));

## control settings for TDMR

```r
tdm <- list(mainfunc="main_sonar"
, runList = c("sonar_04.conf")
, umode="CV" # { "CV" | "SUB" | "TST" | "SP_T" }
, tuneMethod = c("lhd")
, filenameEnvT="exBigLoop.RData" # file to save environment envT (in dir 'path')
, nrun=1, nfold=2 # repeats and CV-folds for the unbiased runs
, nExperim=1
, parallelCPUs=1
, parallelFuncs=c("readCmdSonar")
, optsVerbosity = 0 # the verbosity for the unbiased runs)

## Each element of tdm$runList has the settings for one tuning process (e.g.
## - auto.loop.steps = number of SPOT generations
## - auto.loop.evals = budget of model building runs and
## - io.roi(fileName = "sonar_04.roi"
## ).

spotStep = "auto";
source(paste(path,"start_bigLoop.r",sep="/"),chdir=TRUE);  # change dir to 'path' while sourcing
**tdmBindResponse**  
*Bind a column to a data frame.*

**Description**  
Bind the column with name `response.predict` and contents `vec` as last column to data frame `d`.

**Usage**  
```
tdmBindResponse(d, response.predict, vec)
```

**Arguments**
- `d` data frame
- `response.predict` name of new column
- `vec` the contents for the last column added to data frame `d`

**Value**
- data frame `d` with column added

---

**tdmClassify**  
*Core classification function of TDMR.*

**Description**

`tdmClassify` is called by `tdmClassifyLoop` and returns an object of class `tdmClass`. It trains a model on training set `d_train` and evaluates it on test set `d_test`. If this function is used for tuning, the test set `d_test` plays the role of a validation set.

**Usage**  
```
tdmClassify(d_train, d_test, d_dis, d_preproc,  
            response.variables, input.variables, opts)
```

**Arguments**
- `d_train` training set
- `d_test` validation set, same columns as training set
- `d_dis` 'disregard set', i.e. everything what is neither train nor test. The model is applied to all records in `d_dis` (needed for active learning, see ssl_methods.r)
- `d_preproc` data used for preprocessing. May be NULL, if no preprocessing is done (opts$PRE.SFA=="none" and opts$PRE.PCA=="none"). If preprocessing is done, then `d_preproc` is usually all non-validation data.
response.variables
name of column which carries the target variable - or - vector of names specifying multiple target columns (these columns are not used during prediction, only for evaluation)

input.variables
vector with names of input columns

opts
additional parameters [defaults in brackets]

SRF.* several parameters for tdmModSortedRFimport
RF.* several parameters for RF (Random Forest, defaults are set, if omitted)
SVM.* several parameters for SVM (Support Vector Machines, defaults are set, if omitted)

filename
data.title

MOD.method ["RF"] the main training method ["RF"|"MC.RF"|"SVM"|"NB"]: use [Random Forest| MetaCost-RF| SVM| Naive Bayes] for the main model

MOD.SEED =NULL: get a new random number seed with tdmRandomSeed (different RF trainings).
=any value: set the random number seed to this value (+i) to get reproducible random numbers. In this way, the model training part (RF, NNET, ...) gets always a fixed seed (see also TST.SEED in tdmClassifyLoop)

CLASSWT class weights (NULL, if all classes should have the same weight) (currently used only by methods RF, MC.RF and by tdmModSortedRFimport)

fct.postproc [NULL] name of user-def’d function for postprocessing of predicted output

GD.DEVICE if !="non", then make a pairs-plot of the 5 most important variables and make a true-false bar plot

VERBOSE [2] =2: most printed output, =1: less, =0: no output

Details
Currently d_dis is allowed to be a 0-row data frame, but d_train and d_test must have at least one record.

Value
res, an object of class tdmClass, this is a list containing

d_train training set + predicted class column(s)
d_test test set + predicted class column(s)
d_dis disregard set + predicted class column(s)
avgEVAL list with evaluation measures, averaged over all response variables
allEVAL data frame with evaluation measures, one row for each response variable
lastCmTrain a list with evaluation info for training set (confusion matrix, gain, class errors, ...)
lastCmVali a list with evaluation info for validation set (confusion matrix, gain, class errors, ...)

lastModel the last model built (i.e. for the last response variable)

lastProbs a list with three probability matrices (row: records, col: classes) v_train, v_test, v_dis, if the model provides probabilities; NULL else.

lastPred name of the column where the prediction of the last model is appended to the datasets d_train, d_test and d_dis

predProb a list with two data frames Trn and Val. They contain at least a column IND.dset (index of each train / validation record into data frame dset). If the model has probabilities, then they contain in addition a column for each response variable with the prediction probabilities.

opts parameter list from input, some default values might have been added

The 9 evaluation measures in avgEVAL and allEVAL are cerr.* (misclassification error), gain.* (total gain) and rgain.* (relative gain, i.e. total gain divided by max. achievable gain in *) where * = [trn | tst | tst2] stands for [ training set | test set | test set with special treatment ] and the special treatment is either opts$test2.string = "no postproc" or = "default cutoff".

The five items lastCmTrain, lastCmVali, lastModel, lastProbs, lastPred are specific for the *last* model (the one built for the last response variable in the last run and last fold)

Author(s)
Wolfgang Konen, FHK, Sep'2009 - Jun'2012

See Also
print.tdmClass tdmClassifyLoop tdmRegressLoop

Examples

```r
## This demo shows a simple data mining process (phase 1 of TDMR) for classification on
## dataset iris.
## The data mining process in tdmClassify calls randomForest as the prediction model.
## It is called opts$NRUN=1 time with one random train-validation set splits.
## Therefore data frame res$allEval has one row
##
## opts=tdmOptsDefaultsSet()  # set all defaults for data mining process
gdObj <- tdmGraAndLogInitialize(opts);  # init graphics and log file

data(iris)
response.variables="Species"  # names, not data (!)
input.variables=setdiff(names(iris),"Species")
opts$NRUN=1

idx_train = sample(nrow(iris))[1:110]
iristrain=iris[idx_train,]
iristrain$Species=iris[idx_train,]"Species"
res <- tdmClassify(iristrain,iris[-idx_train,]
numeric(0),)
```
Description

tdmClassifyLoop contains a double loop (opts$NRUN and CV-folds) and calls tdmClassify. It is called by all classification R-functions main_*.
It returns an object of class TDMclassifier.

Usage

```r
tdmClassifyLoop(dset, response.variables, input.variables, opts, tset = NULL)
```

Arguments

dset  the data frame containing training and validation data.
tset  [NULL] If not NULL, this is the validation data set. If NULL, the validation data set is build from dset according to the procedure prescribed in opts$TST.*.
response.variables
name of column which carries the target variable - or - vector of names specifying multiple target columns (these columns are not used during prediction, only for evaluation)
input.variables
vector with names of input columns
opts  a list from which we need here the following entries

NRUN  number of runs (outer loop)
TST.SEED  =NULL: get a new random number seed with tdmRandomSeed. =any value: set the random number seed to this value to get reproducible random numbers and thus reproducible training-test-set-selection. (only relevant in case TST.kind="cv" or "rand") (see also MOD.SEED in tdmClassify)
TST.kind  how to create cvi, handed over to tdmModCreateCVindex. If TST.kind="col", then cvi is taken from dset[,opts$TST.col].
GD.RESTART  [TRUE] =TRUE/FALSE: do/don’t restart graphic devices
GD.DEVICE  ["non""i""win""i""pdf""i""png"]
Value

result, an object of class TDMclassifier, this is a list with results, containing

- lastRes: last run, last fold: result from tdmClassify
- C_train: classification error on training set
- G_train: gain on training set
- R_train: relative gain on training set (percentage of max. gain on this set)
- *_vali: similar, with vali set instead of training set —
- *_vali2: similar, with vali2 set instead of training set —
- Err: a data frame with as many rows as opts$NRUN and 9 columns corresponding to the nine variables described above
- predictions: last run: data frame with dimensions [nrow(dset),length(response.variable)]. In case of CV, all CV predictions (for each record in dset), in other cases mixed validation / train set predictions.
- predProbList: the ith element in this list has data on the prediction probabilities of the ith run. See info on predProb in tdmClassify.

Each performance measure C_*, G_*, R_* is a vector of length opts$NRUN. To be specific, C_train[i] is the classification error on the training set from the i-th run. This error is mean(res$allEVAL$cerr.trn), i.e. the mean of the classification errors from all response variables when res is the return value of tdmClassify. In the case of cross validation, for each performance measure an additional averaging over all folds is done.

Author(s)

Wolfgang Konen (<wolfgang.konen@fh-koeln.de>), FHK, Sep’2010 - Jun’2012

See Also

print.TDMclassifier, tdmClassify, tdmRegress, tdmRegressLoop

Examples

```R
### ........ demo/demo00-0classif.r ........
### This demo shows a simple data mining process (phase 1 of TDMR) for classification on
### dataset iris.
### The data mining process in tdmClassifyLoop calls randomForest as the prediction model.
### It is called opts$NRUN=2 times with different random train-validation set splits.
### Therefore data frame result$Err has two rows
###
### opts=tdmOptsDefaultsSet() # set all defaults for data mining process
gObj <- tdmGraAndLogInitialize(opts); # init graphics and log file
data(iris)
response.variables="Species" # names, not data (!)
input.variables=setdiff(names(iris),"Species")

result = tdmClassifyLoop(iris,response.variables,input.variables,opts)
```
print(result$Err)

dtmClassifySummary

---

**Description**

result$y is "minus OOB rgain" on training set for methods RF or MC.RF. result$y is "minus rgain" on test set (=validation set) for all other methods. result$y is the quantity which the tuner seeks to minimize.

**Usage**

tdmClassifySummary(result, opts, dset = NULL)

**Arguments**

- **result**: return value from a prior call to `tdmClassifyLoop`, an object of class `tdmclassifier`.
- **opts**: a list from which we need here the following entries
  - `Nrun`: number of runs (outer loop)
  - `method`
  - `VERBOSE`
  - `dset`: [NULL] if !=NULL, attach it to result
- **dset**: [NULL] if not NULL, add this data frame to the return value (may cost a lot of memory!)

**Value**

result, an object of class `tdmclassifier`, with `result$y`, `result$sd.y` (and optionally also `result$dset`) added

**Author(s)**

Wolfgang Konen, FHK, Sep’2010 - Oct’2011

**See Also**

tdmClassify, tdmClassifyLoop, print.TDMclassifier, tdmRegressSummary
**tdmDefaultsFill**

**Default values for list tdm. This list controls the tuning and unbiased evaluation phase.**

**Description**

When called with `tdm = tdmDefaultsFill()`, a new list `tdm` is created and returned. When called with `tdm = tdmDefaultsFill(mainFile="my.r")`, a new list `tdm` is created and returned, with the element `mainFile` set to the specified value. When called with `tdm = tdmDefaultsFill(tdm)`, an existing list `tdm` is filled with further default values.

**Usage**

```
tdmDefaultsFill(tdm = NULL, mainFile = NULL)
```

**Arguments**

- `tdm` (optional)
- `mainFile` (optional) if given, create or overwrite `tdm$mainFile` with this value

**Details**

If `tdm$mainFunc` is missing, but `tdm$mainFile` exists, then `tdmDefaultsFill` will set

```
tdm$mainFunc = sub(".r"", ",basename(tdm$mainFile), fixed=TRUE)
```

**Value**

`tdm` the new / extended list, where additional elements, if they are not yet def'd, are set as:

- `mainFile` [NULL] if not NULL, source this file from the current dir. It should contain the definition of `tdm$mainFunc`.
- `mainFunc` `sub(".r"", ",basename(tdm$mainFile), fixed=TRUE)`, if `tdm$mainFile` is set and `tdm$mainFunc` is NULL, else "mainFunc" This is the name of the function called in `tdmStartSpot` and `unbiasedRun`
- `unbiasedFunc` ["unbiasedRun"] which function to call for unbiased evaluation
- `tuneMethod` ["spot"] other choices: "cmaes", "bfgs", ..., see `tdmDispatchTuner`
- `nExperim` [1]
- `umode` ["RSUB"], one out of [ "RSUB" | "CV" | "TST" | "SP_T" ], see `unbiasedRun`
- `timeMode` [1] 1: proc time, 2: system time, 3: elapsed time (columns `timeNtst` and `timeNtrn` in `envT$theFinals`)
- `fileMode` [FALSE] see "Note" section in `tdmBigLoop`
- `finalFile` [NULL] filename where to save `envT$theFinals`, only relevant for `tdm$fileMode==TRUE`
- `experFile` [NULL] filename where to append `envT$theFinals`, only relevant for `tdm$fileMode==TRUE`
filenameEnvT  [NULL] filename where tdmBigLoop will save a small version of environment envT. If NULL, save EnvT to sub(".conf",".RData",tdm$runList[1]). This RData file is written irrespective of fileMode's value, but only in case spotStep="auto".

call.spotPath  [NA] use SPOT's package version

parallelCPUs  [1] 1: sequential, >1: parallel execution with this many CPUs (package parallel)

parallelFuncs  [NULL] in case tdm$parallelCPUs>1: a string vector with functions which are clusterExport'ed in addition to tdm$mainFunc.

path  [NULL] where to search .conf and .apd file. If it is NULL, path is set to the actual working directory at the time when tdmEnvTMakeNew is executed

runList  [NULL] a list of .conf filenames

spotList  [NULL] a list of .conf filenames

stratified  [NULL] see tdmSplitTestData

tdmPath  [NULL] from where to source the R sources. If NULL load library TDMR instead.

testRNstring  ["default cutoff"]

opts$verbosity  [0] the verbosity for the unbiased runs

withParams  [TRUE] list the columns with tuned parameter in final results

nrun  [5] number of runs for unbiased run

U.saveModel  [FALSE] if TRUE, save the last model, which is trained in unbiasedRun, onto filenameEnvT

tstCol  ["TST"] opts$TST.COL for unbiased runs (only for umode="TST")
nfold  [10] number of CV-folds for unbiased runs (only for umode="CV")

TST.trnFrac  [NULL] train set fraction (of all train-vali data), OVERWRITES opts$TST.trnFrac if not NULL.

TST.valiFrac  [NULL] validation set fraction (of all train-vali data), OVERWRITES to opts$TST.valiFrac if not NULL.

TST.testFrac  [0.2] test set fraction (of *all* data) for unbiased runs (only for umode="RSUB" or="SP_T")

CMA.propertyFile  [NULL] (only for CMA-ES Java tuner) see cma_jTuner.

CMA.populationSize  [NULL] (only for CMA-ES Java tuner) see cma_jTuner.

Note

The settings tdm$TST.trnFrac and tdm$TST.valiFrac allow to set programmatically certain values for opts$TST.trnFrac and opts$TST.valiFrac *after* opts has been read from APD file. So use tdm$TST.trnFrac and tdm$TST.valiFrac with CAUTION!

For tdm$timeMode, the 'user time' is the CPU time charged for the execution of user instructions of the calling process. The 'system time' is the CPU time charged for execution by the system on behalf of the calling process. The 'elapsed time' is the 'real' (wall-clock) time since the process was started.
Author(s)

Wolfgang Konen, Patrick Koch, 2011 - 2013

tdmEmbedDataFrame

Embed columns in a data frame.

Description

Embeds the columns named cols in the data frame x into a space of dimension dimension.

Usage

```
tdmEmbedDataFrame(x, cols = NULL, dimension = 1, take = 1)
```

Arguments

- **x**
  - The data frame containing the columns to embed.
- **cols**
  - [NULL] A vector with a list of names of the columns to embed. If NULL, embed all columns of x.
- **dimension**
  - [1] The additional dimensions to generate when embedding.
- **take**
  - [1] Take from these dimensions only every take'th column.

Value

The data frame, augmented with embedded columns, shortened by dimension rows.

tdmEnvTAddBstRes

Add BST and RES data frames to an existing envT environment.

Description

Load an envT-type environment from file fileRData. Its elements bst, bstGrid res, and resGrid overwrite the elements in envT passed in as argument.

Usage

```
tdmEnvTAddBstRes(envT, fileRData)
```

Arguments

- **envT**
  - the TDMR environment
- **fileRData**
  - string with filename to load. This file is searched in envT$tdm$path.

Value

the augmented envT
Add getter functions `getBst` and `getRes` to environment `envT`.

**tdmEnvTAddGetters**

**Description**
Add getter functions `getBst` and `getRes` to environment `envT`.

**Usage**
```
tdmEnvTAddGetters(envT)
```

**Arguments**
- `envT`: the TDMR environment

**Value**
- the augmented `envT`

Load an `envT`-type environment from file `fileRData`.

**tdmEnvTLoad**

**Description**
The loaded `envT` is augmented with getter functions, see `tdmEnvTAddGetters`.

**Usage**
```
tdmEnvTLoad(fileRData)
```

**Arguments**
- `fileRData`: string with filename to load. This file is searched in the working dir.

**Value**
- `envT`
Construct a new environment envT of class TDMenvir.

Description

Given the general TDMR settings in tdm, construct an appropriate environment envT. This is needed as input for tdmBigLoop.

Usage

```r
tdmEnvTMakeNew(tdm = NULL)
```

Arguments

- `tdm`: a list with general settings for TDMR, see tdmDefaultsFill

Value

Environment envT, an object of class TDMenvir, containing (among others) the elements

- `runList` = tdm$runList
- `spotList` = tdm$spotList
- `tdm` = tdmDefaultsFill(tdm)
- `getBst` accessor function(confFile,nExp,theTuner) into envT$bstGrid
- `getRes` accessor function(confFile,nExp,theTuner) into envT$resGrid
- `sCList` list of spotConfig-objects, as many as envT$runList has elements. Each spotConfig object sCList[[k]] contains a list opts as element, which is read from .apd file specified in envT$runList[k].

Author(s)

Wolfgang Konen (<wolfgang.konen@fh-koeln.de>), Patrick Koch

See Also

tdmBigLoop
Re-read APD-files (list opts) for an environment envT of class TDMenvir.

**Description**

Given an already existing envT with envT$sClist filled, re-read the opts settings from APD files whose filenames are specified in envTSsList[[k]]$io.apfilename.

**Usage**

```r
tdmEnvTReadApd(envT, tdm)
```

**Arguments**

- `envT` an environment of class `TDMenvir`
- `tdm` a list with general settings for TDMR (can be envT$tdm)

**Value**

modified environment envT, an object of class `TDMenvir`, where the items `sCList[[k]]$opts = tdm$runList` are modified

**Author(s)**

Wolfgang Konen (<wolfgang.konen@fh-koeln.de>), Patrick Koch

**See Also**

- `tdmEnvTMakeNew`

---

Make a sensitivity plot based on envT

**Description**

Given the results from a prior tuning run in envT, make a sensitivity plot for this run (call SPOT with spotStep="rep") If envT$tdm$run > 0 then make additionally with the best-performing parameters from the tuning run a new unbiased run on the test data.

**Usage**

```r
tdmEnvTSensi(envT, ind)
```
tdmExecSpotStep

Arguments

envT results from a prior tuning run. If envT$tdm$run==0
ind an integer from 1:length(envT$bstGrid): Take the tuning run with index ind.

Value

finals, either NULL (no unbiased run) or a one-row data frame with the results from the unbiased run

Examples

## The best results are read from demo02sonar/demoSonar.RData relative to the TDMR package directory.
oldwd <- getwd(); setwd(paste(find.package("TDMR"), "demo02sonar", sep="/"));
envT = tdmEnvTLoad("demoSonar.RData"); # loads envT
source("main_sonar.r");
envT$tdm$run=0; # =0: don't, >0: do unbiasedRun with opts$NRUN=envT$tdm$run
finals = tdmEnvTSensi(envT,1);
if (!is.null(finals)) print(finals);
setwd(oldwd);

---

tdmExecSpotStep Execute the tuning process with the given spotStep

Description

If spotStep="auto", construct envT from the general TDMR settings in tdm.
If spotStep="rep" or "report", load envT from file tdm$filenameEnvT (from a prior tuning run).
Then call tdmBigLoop(envT,spotStep).

Usage

tdmExecSpotStep(tdm, spotStep)

Arguments

tdm a list with general settings for TDMR, see tdmDefaultsFill
spotStep a string, either "auto", "rep" or "report"

Details

If tdm$filenameEnvT is NULL, set it to sub(".conf",".RData",tdm$runList[1]).
Value

Environment `envT` containing (among others) the elements

- `runList` = `tdm$runList`
- `spotList` = `tdm$spotList`
- `tdm` = `tdm`
- `getBst` accessor function(`confFile`, `nExp`, `theTuner`) into `envT$bstGrid`
- `getRes` accessor function(`confFile`, `nExp`, `theTuner`) into `envT$resGrid`
- `sCList` list of `spotConfig` objects, as many as `envT$runList` has elements. Each `spotConfig` object `sList[[k]]` contains a list `opts` as element, which is read from `.apd` file specified in `envT$runList[k]`.

Author(s)

Wolfgang Konen (<wolfgang.konen@fh-koeln.de>)

See Also

`tdmBigLoop`, `tdmEnvTMakeNew`

tdmGraAndLogFinalize  Finalize graphics and log file

Description

Finalize graphics and log file

Usage

```r
tdmGraAndLogFinalize(opts, gdObj = NULL)
```

Arguments

- `opts` with `opts$GD.DEVICE` one out of ["pdf" | "png" | "win" | "rstudio" | "non"], see `tdmGraphicInit`
- `gdObj` object of class `TDMgdev`, the return value from `tdmGraAndLogInitialize`, to ensure that `tdmGraAndLogInitialize` was called before (and the sink on `opts$LOGFILE` can be closed)
tdmGraAndLogInitialize

Initialize graphics and log file.

Description

The log file is opened in opts$dir.output.opts$logfile, but only if opts$fileMode==TRUE.

Usage

tdmGraAndLogInitialize(opts)

Arguments

opts with opts$GD.DEVICE one out of ["pdf" | "png" | "win" | "rstudio" | "non"], see tdmGraphicInit

Value

gdObj, an object of class TDMgdev. Pass this object on when calling tdmGraAndLogFinalize(opts, gdObj) (if not, a warning is issued before the sink-closing-error occurs)

tdmGraphicCloseDev

Close all open graphic devices.

Description

Close all open graphic devices.

Usage

tdmGraphicCloseDev(opts, ...)

Arguments

opts with opts$GD.DEVICE one out of ["pdf" | "png" | "win" | "rstudio" | "non"], see tdmGraphicInit

... optional arguments (currently not used)
### tdmGraphicCloseWin

*Close active file ("png").*

**Description**

Close active file ("png").

**Usage**

```
tdmGraphicCloseWin(opts, ...)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opts</td>
<td>with <code>opts$GD.DEVICE</code> one out of [&quot;pdf&quot;</td>
</tr>
<tr>
<td>...</td>
<td>optional arguments (currently not used)</td>
</tr>
</tbody>
</table>

### tdmGraphicInit

*Initialize graphic device.*

**Description**

Open multipage PDF or (create and) clear `opts$GD.PNGDIR`.

**Usage**

```
tdmGraphicInit(opts, ...)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opts</td>
<td>with <code>opts$GD.DEVICE</code> one out of [&quot;pdf&quot;</td>
</tr>
<tr>
<td>...</td>
<td>optional arguments to hand over to <code>pdf</code> (the other devices require no further arguments)</td>
</tr>
</tbody>
</table>

"pdf" plot everything in one multipage pdf file `opts$PDFFILE`

"png" each plot goes into a new png file in `opts$GD.PNGDIR`

"win" each plot goes into a new window (dev.new())

"rstudio" plot everything to the RStudio plot device (has a history)

"non" all plots are suppressed
**tdmGraphicNewWin**  

Initialize a new window.

---

**Description**

Initialize a new window ("win") / a new file ("png") for current graphic device.

**Usage**

```r
tdmGraphicNewWin(opts, ...)
```

**Arguments**

- `opts`  
  with `opts$GD.DEVICE` one out of ["pdf" | "png" | "win" | "rstudio" | "non"], see `tdmGraphicInit`

- `...`  
  optional arguments to hand over to `png` or `windows` or `X11` in package `grDevices` (the other devices require no further arguments)

---

**tdmGraphicToTop**  

Bring the active window to the top

---

**Description**

Only relevant for `opts$GD.DEVICE=="win"`.

**Usage**

```r
tdmGraphicToTop(opts)
```

**Arguments**

- `opts`  
  with `opts$GD.DEVICE` one out of ["pdf" | "png" | "win" | "rstudio" | "non"], see `tdmGraphicInit`
tdmMapDesApply

Apply the mapping from des to opts.

**Description**

For each variable which appears in .roi (and thus in .des file and design point data frame des): set its counterpart in list opts to the values of the k-th row in des. For each variable not appearing: leave its counterpart in opts at its default value from .apd file.

**Usage**

```r
tdmMapDesApply(des, opts, k, spotConfig, tdm)
```

**Arguments**

- `des`  
  design points data frame
- `opts`  
  list of options
- `k`  
  apply mapping for the k-th design point
- `spotConfig`  
  list, we needed here spotConfig$alg.roi and envTormapUser, see `tdmMapDesLoad`, and in addition envT$spotConfig$alg.roi
- `tdm`  
  list, we need here tdm$map and tdm$mapUser

**Value**

opts, the modified list of options

**See Also**

`tdmMapDesLoad`

---

**tdmMapDesLoad**

Load the mapping files.

**Description**

Load the map files "tdmMapDesign.csv" and optionally also "userMapDesign.csv" and store them in tdm$map and tdm$mapUser, resp. These maps are used by `tdmMapDesApply`. "tdmMapDesign.csv" is searched in the TDMR library path `find.package("TDMR")`. (For the developer version: `<tdm$tdmPath>/inst`). "userMapDesign.csv" is searched in tdm$path (which is getwd() if the user did not define tdm$path).

**Usage**

```r
tdmMapDesLoad(tdm = list())
```
tdmModConfmat

Arguments

  tdm         list, needed for tdm$tdmPath and tdm$path

Value

  tdm, the modified list with new elements tdm$map and tdm$mapUser

See Also

  tdmMapDesApply

Description

Calculate confusion matrix, gain and RGain measure.

Usage

  tdmModConfmat(d, colreal, colpred, opts, predProb = NULL)

Arguments

  d         data frame
  colreal   name of column in d which contains the real class
  colpred   name of column in d which contains the predicted class
  opts      a list from which we use the elements:
            • gainmat: the gain matrix for each possible outcome, same size as cm$mat
              (see below).
              gainmat[R1,P2] is the gain associated with a record of real class R1 which
              we predict as class P2. (gain matrix = - cost matrix)
            • rgain.type: one out of {"rgain" | "meanCA" | "minCA" | "arROC" | "arLIFT" | "arPRE" }, affects output cm$mat and cm$rgain, see below.
  predProb  if not NULL, a data frame with as many rows as data frame d, containing
            columns (index, true label, predicted label, prediction score). Is only needed
            for opts$rgain.type="arx".
Value

- **cm**, a list containing:
  - **mat**: matrix with real class levels as rows, predicted class levels columns. 
    \( \text{mat}[R1, P2] \) is the number of records with real class R1 predicted as class P2, if \( \text{opts$rgain.type} == "\text{rgain}" \). If \( \text{opts$rgain.type} == "\text{meanCA}" \) or "minCA", then show this number as percentage of "records with real class R1" (percentage of each row). CAUTION: If colpred contains NA’s, those cases are missing in mat (!) (but the class errors are correct as long as there are no NA’s in colreal).
  - **cerr**: class error rates, vector of size \( \text{nlevels(colreal)} + 1 \). 
    \( \text{cerr}[X] \) is the misclassification rate for real class X. 
    \( \text{cerr["Total"]} \) is the total classification error rate.
  - **gain**: the total gain (sum of pointwise product \( \text{opts$gainmat*cm$mat} \))
  - **gain.vector**: \( \text{gain.vector}[X] \) is the gain attributed to real class label X. \( \text{gain.vector["Total"]} \) is again the total gain.
  - **gainmax**: the maximum achievable gain, assuming perfect prediction
  - **rgain**: ratio gain/gainmax in percent, if \( \text{opts$rgain.type} == "\text{rgain}" \); 
    mean class accuracy percentage (i.e. mean(diag(cm$mat)), if \( \text{opts$rgain.type} == "\text{meanCA}" \); 
    min class accuracy percentage (i.e. min(diag(cm$mat)), if \( \text{opts$rgain.type} == "\text{minCA}" \); 
    area under ROC curve (a number in [0,1]), if \( \text{opts$rgain.type} == "\text{arROC}" \); 
    area between lift curve and horizontal line 1.0, if \( \text{opts$rgain.type} == "\text{arLIFT}" \); 
    area under precision-recall curve (a number in [0,1]), if \( \text{opts$rgain.type} == "\text{arPRE}" \);

Author(s)

Wolfgang Konen (<wolfgang.konen@fh-koeln.de>), Patrick Koch

See Also

- **tdmClassify**
- **tdmROCRbase**

---

Description

Depending on the value of member TST.kind in list opts, the returned index cvi is

1. **TST.kind="cv"**: a random cross validation index \( P([111...222...333...]) \) - or -
2. **TST.kind="rand"**: a random index with \( P([00...11...-1-1...]) \) for training (0), validation (1) and disregard (-1) cases - or -
3. **TST.kind="col"**: the column dset[,opts$TST.COL] contains the training (0), validation (1) and disregard (-1) set division (and all records with a value <0 in column TST.COL are disregarded).
Here \( P(.) \) denotes random permutation of the sequence. The disregard set is optional, i.e. cvi may contain only 0 and 1, if desired. Special case TST.kind="cv" and TST.NFOLD=1: make *every* record a training record, i.e. index [000...]. In case TST.kind="rand" and stratified=TRUE a stratified sample is drawn, where the strata in the training case reflect the rel. frequency of each level of the *1st* response variable and are ensured to be at least of size 1.

Usage

```r
tdmModCreateCVindex(dset, response.variables, opts, stratified = FALSE)
```

Arguments

dset the data frame for which cvi is needed
response.variables issue a warning if length(response.variables)>1. Use the first response variable for determining strata size.

opts a list from which we need here the following entries

- TST.kind: ["cv"|"rand"|"col"]
- TST.NFOLD: number of CV folds (only relevant in case TST.kind=="cv")
- TST.COL: column of dset containing the (0/1/<0) index (only relevant in case TST.kind=="col") or NULL if no such column exists
- TST.valiFrac: fraction of records to set aside for validation (only relevant in case TST.kind=="rand")
- TST.trnFrac: [1-opts$TST.valiFrac] fraction of records to use for training (only relevant in case TST.kind=="rand")

stratified [F] do stratified sampling for TST.kind="rand" with at least one training record for each response variable level (classification)

Value
cvi training-validation-set (0/>0) index vector (all records with cvi<0, e.g. from column TST.COL, are disregarded)

Note
Currently stratified sampling in case TST.KIND='rand' does only work correctly for one response variable. If there are more than one, the right fraction of validation records is taken, but the strata are drawn w.r.t. the first response variable. (For multiple response variables we would have to return a list of cvi's or to call tdmModCreateCVindex for each response variable anew.)
Sort the input variables decreasingly by their RF-importance.

Description

Build a Random Forest using importance=TRUE. Usually the RF is smaller (50 trees), to speed up computation. Use na.roughfix for missing value replacement. Decide which input variables to keep and return them in SRF$input.variables

Usage

tdmModSortedRfimport(d_train, response.variable, input.variables, opts)

Arguments

d_train training set
tdmsortedrfimport() as above
response.variable the target column from d_train to use for the RF-model
tdmModSortedRfimport() as above
input.variables the input columns from d_train to use for the RF-model
tdmModSortedRfimport() as above
opts options, here we use the elements [defaults in brackets]:

• SRF.kind:
  ="xperc": keep a certain importance percentage, starting from the most important variable
  ="ndrop": drop a certain number of least important variables
  ="nkeep": keep a certain number of most important variables
  ="none": do not call tdmModSortedRfimport at all (see tdmRegress.r and tdmClassify.r)

• SRF.ndrop: [0] how many variables to drop (if SRF.kind=="ndrop")

• SRF.XPerc: [0.95] if >=0, keep that importance percentage, starting with
  the most important variables (if SRF.kind=="xperc")

• SRF.calc: [TRUE] =TRUE: calculate importance & save on SRF.file, =F:
  load from SRF.file (SRF.file = Output/<filename>.SRF.<response.variable>.Rdata)

• SRF.ntree: [50] number of RF trees

• SRF.verbose: [2]

• SRF.maxS: [40] how many variables to show in plot

• SRF.minS: [1] a lower bound for the length of SRF$input.variables

• RF.sampsize: sampsize for RF, set prior to calling this func via tdmModAd-
  justSampsize(opts$SRF.samp,...)

• GD.DEVICE: if !="non", then make a bar plot on current graphic device

• CLS.CLASSWT: class weight vector to use in random forest training
Value

SRF, a list with the following elements:
- **input.variables**
  - the vector of input variables which remain after importance processing. These are sorted by decreasing importance.
- **s_input**
  - all input variables sorted by decreasing (**NEW**) importance
- **s_imp1**
  - the importance for s_input
- **s_dropped**
  - vector with name of dropped variables
- **lsd**
  - length of s_dropped
- **perc**
  - the percentage of total importance which is in the dropped variables
- **opts**
  - some defaults might have been added

Author(s)

Wolfgang Konen, Patrick Koch <wolfgang.konen@fh-koeln.de>

---

**tdmModVote2Target**  
*Analyze how the vote fraction corresponds to reliability of prediction.*

Description

This function analyzes whether in different vote bins the trained RF makes predictions with different reliability. Only for RF-prediction in case of binary (0/1) classification.

Expected result: The larger the fraction of trees voting for class 0 is, the smaller is the percentage of true class-1 cases in this vote bin. This function is somewhat specialized for the DMC2010-task.

Usage

`tdmModVote2Target(vote, pred, target)`

Arguments

- **vote**
  - vector: which fraction of trees votes for class 0?
- **pred**
  - vector: the predicted class for each record (0/1)
- **target**
  - vector: the true class for each vector (0/1)

Value

a data frame with columns

- **vcut**
  - vote cut v
- **count**
  - number of cases with vote fraction in [v[i-1],v[i]]
- **pred**
  - fraction of 0-predictions
- **pCorr**
  - fraction of correct predictions
- **pR**
  - fraction of true 1-cases
tdmOptsDefaultsSet

Default values for list opts.

Description

Set up and return a list opts with default settings. The list opts contains all DM-related settings which are needed by main_<TASK>.

For better readability, most elements of opts are arranged in groups:

- **dir.** path-related settings
- **READ.** data-reading-related settings
- **TST.** resampling-related settings (training, validation and test set, CV)
- **PRE.** preprocessing parameters
- **SRF.** several parameters for tdmModSortedRFimport
- **MOD.** general settings for models and model building
- **RF.** several parameters for model RF (Random Forest)
- **SVM.** several parameters for model SVM (Support Vector Machines)
- **ADA.** several parameters for model ADA (AdaBoost)
- **CLS.** classification-related settings
- **GD.** settings for the graphic devices

Usage

```
  tdmOptsDefaultsSet(opts = NULL, path = "/")
```

Arguments

- **opts** (optional) the options already set
- **path** ["./"] the path string to precede all settings opts$dir.*. If path="./" then nothing is preceeded to opts$dir.*.

Details

The path-related settings are relative to dir(tdm$mainFile), if it is def’d, else relative to the current dir.
Finally, the function tdmOptsDefaultsFill(opts) is called to fill in further details, depending on the current settings of opts.

Value

a list opts, with defaults set for all options relevant for a DM task, containing the following elements
tdmOptsDefaultsSet

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dir.txt</td>
<td><code>&lt;path&gt;/data</code> where to find .txt/.csv files</td>
</tr>
<tr>
<td>dir.data</td>
<td><code>&lt;path&gt;/data</code> where to find other data files, including .Rdata</td>
</tr>
<tr>
<td>dir.Rdata</td>
<td><code>&lt;path&gt;/Rdata</code> – deprecated, use opts$dir.data –</td>
</tr>
<tr>
<td>dir.output</td>
<td><code>&lt;path&gt;/Output</code> where to put output files</td>
</tr>
<tr>
<td>filename</td>
<td>[&quot;default.txt&quot;] the task data</td>
</tr>
<tr>
<td>filetest</td>
<td>[NULL] the test data, only relevant for READ.TST=T</td>
</tr>
<tr>
<td>fileMode</td>
<td>[TRUE] if =T, write opts$EVALFILE=*_train_eval.csv, <em>_train.csv.SRF.</em>, .RData file and *_train.log file</td>
</tr>
<tr>
<td>logfile</td>
<td>[TRUE] if =T, and if opts$fileMode=T, write log file to *_train.log</td>
</tr>
<tr>
<td>data.title</td>
<td>[&quot;Default Data&quot;] title for plots</td>
</tr>
<tr>
<td>READ.NROW</td>
<td>[-1] read this amount of rows or -1 for 'read all rows'</td>
</tr>
<tr>
<td>READ.TST</td>
<td>[F] =T: read unseen test data from opts$filetest (usually you will do this only for the final model and only with TST.kind=&quot;col&quot;)</td>
</tr>
<tr>
<td>READ.CMD</td>
<td>[&quot;tdmReadCmd(filename, opts)&quot;] the command to be passed into tdmReadData. It has to contain the placeholder 'filename'. The default in brackets implies 'read.csv(file=paste(opts$dir.txt, filename, sep=&quot;&quot;), nrow=opts$READ.NROW)' which includes the further settings header=T, sep=&quot;&quot;, and dec=&quot;&quot;.</td>
</tr>
<tr>
<td>READ.INI</td>
<td>[TRUE] read the task data initially, i.e. prior to tuning, using tdmReadData. If =FALSE, the data are read anew in each pass through main_TASK, i.e. in each tuning step (deprecated).</td>
</tr>
<tr>
<td>TST.kind</td>
<td>[&quot;rand&quot;] one of the choices from {&quot;cv&quot;,&quot;rand&quot;,&quot;col&quot;}, see tdmModCreateCVindex for details</td>
</tr>
<tr>
<td>TST.COL</td>
<td>[&quot;TST.COL&quot;] name of column with train/test/disregard-flag</td>
</tr>
<tr>
<td>TST.NFOLD</td>
<td>[3] number of CV-folds (only for TST.kind==&quot;cv&quot;)</td>
</tr>
<tr>
<td>TST.valiFrac</td>
<td>[0.1] set this fraction of the train-validation data aside for validation (only for TST.kind==&quot;rand&quot;)</td>
</tr>
<tr>
<td>TST.testFrac</td>
<td>[0.1] set prior to tuning this fraction of data aside for testing (if tdm$umode==&quot;SP_T&quot; and opts$READ.INI==TRUE) or set this fraction of data aside for testing after tuning (if tdm$umode==&quot;RSUB&quot; or ==&quot;CV&quot;)</td>
</tr>
<tr>
<td>TST.trnFrac</td>
<td>[NULL] train set fraction, if NULL then tdmModCreateCVindex will set it to 1 - opts$TST.valiFrac.</td>
</tr>
<tr>
<td>TST.SEED</td>
<td>[NULL] a seed for the random test set selection (tdmRandomSeed) and random validation set selection. (tdmClassifyLoop). If NULL, use tdmRandomSeed.</td>
</tr>
<tr>
<td>PRE.PCA</td>
<td>[&quot;none&quot; (default)</td>
</tr>
<tr>
<td>PRE.PCA.REPLACE</td>
<td>[T] =T: replace with the PCA columns the original numerical columns, =F: add the PCA columns</td>
</tr>
<tr>
<td>PRE.PCA.npc</td>
<td>[0] if &gt;0: add monomials of degree 2 from the first PRE.PCA.npc columns (PCs) (only active, if opts$PRE.PCA!=&quot;none&quot;)</td>
</tr>
</tbody>
</table>
PRE.SFA: ["none" (default)|"2nd"] SFA preprocessing (see package rSFA: [don’t l do ormal SFA with 2nd degree expansion ]

PRE.SFA.REPLACE

[F] =T: replace the original numerical columns with the SFA columns; =F: add the SFA columns

PRE.SFA.npc

[0] if >0: add monomials of degree 2 from the first PRE.SFA.npc columns (only active, if opts$PRE.SFA!="none")

PRE.SFA.PPRANGE

[11] number of inputs after SFA preprocessing, only those inputs enter into SFA expansion

PRE.SFA.ODIM

[5] number of SFA output dimensions (slowest signals) to return

PRE.SFA.doPB

[T] =F: don’t l do parametric bootstrap for SFA in case of marginal training data

PRE.SFA.fctPB

[sfaPBootstrap] the function to call in case of parametric bootstrap, see sfaPBootstrap in package rSFA for its interface description

PRE.allNonVali

[F] if =T, then use all non-validation data in the training-validation set for PCA or SFA preprocessing. If =F, use only the training set for PCA or SFA processing (only relevant if opts$PRE.PCA!="none" or opts$PRE.SFA!="none")

PRE.Xpgroup

[0.99] bind the fraction 1-PRE.Xpgroup in column OTHER (see tdmPreGroupLevels)

PRE.MaxLevel

[32] bind the N-32+1 least frequent cases in column OTHER (see tdmPreGroupLevels)

SRF.kind

["xperc" (default) |"ndrop" |"nkeep" |"none" ] the method used for feature selection, see tdmModSortedRFImport

SRF.ndrop

[0] how many variables to drop (only relevant if SRF.kind=="ndrop")

SRF.nkeep

[NULL] how many variables to keep, NULL="keep all" (only relevant if SRF.kind=="nkeep")

SRF.XPerc

[0.95] if >=0, keep that importance percentage, starting with the most important variables (if SRF.kind=="xperc")

SRF.calc


SRF.ntree

[50] number of RF trees

SRF.samp

sampsize for RF in importance estimation. See RF.samp for further info on sampsize.

SRF.verbose

[2]

SRF.maxS

[40] how many variables to show in plot

SRF.minlsi

[1] a lower bound for the length of SRFInput.variables

SRF.method

["RFimp"]

SRF.scale

[TRUE] option ‘scale’ for call importance() in tdmModSortedRFImport

MOD.SEED

[NULL] a seed for the random model initialization (if model is non-deterministic). If NULL, use tdmRandomSeed.

MOD.method

["RF" (default) |"MC.RF" |"SVM" |"NB" ]: use [RF | MetaCost-RF | SVM | Naive Bayes] in tdmClassify
["RF" (default) |"SVM" |"LM" ]: use [RF | SVM | linear model ] in tdmRegress
RF.ntrtree [500]
RF.samp [NULL] sampsize for RF in model training. If RF.samp is a scalar, then it specifies the total size of the sample. For classification, it can also be a vector of length n.class (= # of levels in response variable), then it specifies the size of each strata. The sum of the vector is the total sample size. If NULL, RF.samp will be replaced by 3000 later in tdmModAdjSampsize*.
RF.mtry [NULL]
RF.nnodesize [1]
RF.OOB [TRUE] if =T, return OOB-training set error as tuning measure; if =F, return validation set error
RF.p.all [TRUE] if =T, return OOB-training set error as tuning measure; if =F, return validation set error
SVM.epsilon [0.005] needed only for regression
SVM.gamma [0.005]
SVM.coef0 [0.0] (needed only for opts$SVM.kernel=="polynomial" or =="sigmoid")
SVM.degree [3] (needed only for opts$SVM.kernel=="polynomial")
SVM.tolerance [0.008]
ADA.mfinal [10] number of trees in AdaBoost = mfinal boosting(...,mfinal,...)
ADA.rpart.minsplit [20] minimum number of observations in a node in order for a split to be attempted
CLS.cutoff [NULL] vote fractions for the classes (vector of length n.class = # of levels in response variable). The class i with maximum ratio (% votes)/CLS.cutoff[i] wins. If NULL, then each class gets the cutoff 1/n.class (i.e. majority vote wins). The smaller CLS.cutoff[i], the more likely class i will win.
CLS.CLASSWT [NULL] class weights for the n.class classes, e.g. c(A=10,B=20) for a 2-class problem with classes A and B (the higher, the more costly is a misclassification of that real class). It should be a named vector with the same length and names as the levels of the response variable. If no names are given, the levels of the response variables in lexicographical order will be attached in tdmClassify. CLS.CLASSWT=NULL for no weights.
CLS.gainmat [NULL] (n.class x n.class) gain matrix. If NULL, CLS.gainmat will be set to unit matrix in tdmClassify
rgain.type ["rgain" (default) ] in case of tdmClassify: For classification, the measure Rgain returned from tdmClassifyLoop in result$R_* is [relative gain (i.e. gain/gainmax) | mean class accuracy | minimum class accuracy]. The goal is to maximize Rgain.
For binary classification there are the additional measures ["arROC" | "arLIFT" | "arPRE"], see tdmModConfmat.
For regression, the goal is to minimize result$R_* returned from tdmRegress.
In this case, possible values are `rgain.type = ["rmae" (default) | "rmse" | "made"]` which stands for [relative mean absolute error | root mean squared error | mean absolute deviation].

ncopies [0] if >0, activate `tdmParaBootstrap` in `tdmClassify`

fct.postproc [NULL] name of a function with signature (pred, dframe, opts) where pred is the prediction of the model on the data frame dframe and opts is this list. This function may do some postprocessing on pred and it returns a (potentially modified) pred. This function will be called in `tdmClassify` if it is not NULL.

GD.DEVICE ['"win"'] = "win": all graphics to (several) windows (windows or X11 in package grDevices) = "rstudio": same as "win", but all graphics go to the RStudio device = "pdf": all graphics to one multi-page PDF = "png": all graphics in separate PNG files in opts$GD.pngdir = "non": no graphics at all

This concerns the TDMR graphics, not the SPOT (or other tuner) graphics. If running R from RStudio (if there is a device with name "RStudioGD") then the default "win" is changed to "rstudio" automatically.

GD.RESTART [T] = T: restart the graphics device (i.e. close all 'old' windows or re-open multi-page pdf) in each call to `tdmClassify` or `tdmRegress`, resp.
= F: leave all windows open (suitable for calls from SPOT) or write more pages in same pdf.

GD.CLOSE [T] = T: close graphics device "png", "pdf" at the end of main_*._r (suitable for main_*._r solo) or
= F: do not close (suitable for call from tdmStartSpot, where all windows should remain open)

NRun [2] how many runs with different train & test samples - or - how many CV-runs, if opts$TST.kind="cv"

APPLY.TIME [FALSE]
test2.show [FALSE]
test2.string ["default cutoff"]
VERIFY [2] = 2: print much output, = 1: less, = 0: none

**Note**

The variables opts$PRE.PCA.numericV and opts$PRE.SFA.numericV (string vectors of numeric input columns to be used for PCA or SFA) are not set by `tdmOptsDefaultsSet` or `tdmOptsDefaultsFill`. Either they are supplied by the user or, if NULL, TDMR will set them to `input.variables` in `tdmClassifyLoop`, assuming that all columns are numeric.

**Author(s)**

Wolfgang Konen, FHK, 2011 - 2013

**See Also**

`tdmOptsDefaultsFill` `tdmDefaultsFill`
Parametric bootstrap: add 'noisy copies' to a data frame (training data).

Description

A normal distribution is approximated from the data given in dset[, input.variables] and new data are drawn from this distribution for the columns input.variables. The column resp is filled at random with levels with the same relative frequency as in dset[, resp]. Other columns of dset are filled by copying the entries from the first row of dset.

Usage

tdmParaBootstrap(dset, resp, input.variables, opts)

Arguments

dset       data frame with training set
resp       name of column in dset which carries the target variable
input.variables     vector with names of input columns
opts       additional parameters [defaults in brackets]
ncopies   how many noisy copies to add
ncsigma   [1.0] multiplicative factor for each std.dev.
ncmethod   [3] which method to use for parametric bootstrap
   =1: each 'old' record from X in turn is the centroid for a new pattern;
   =2: the centroid is the average of all records from the same class, the std.dev. is the same for all classes;
   =3: centroid as in '2', the std.dev. is the std.dev. of all records from the same class (*recommended*)
TST.COL (optional) name of column in dset where each PB record is marked with a 0

Value

data frame dset with the new parametric bootstrap records added as last rows.

Author(s)

Wolfgang Konen, FHK, Nov’2011-Dec’2011

See Also

tdmClassify
tdmPlotResMeta

Interactive plots of RES data frames and their metamodels.

Description

Makes interactive plots for any of the result data frames contained in envT together with fitted metamodels. tdmPlotResMeta creates a twiddle interface which allows to select

- tuner: one of the tuners
- nExperim: a knob to select one of the experiments in envT (only if envT$tdm$nExperim > 1)
- reportFunc: one of "spotReport3d", "spotReportContour", see SPOT
- modelFit: one of "spotPredictGausspr", "spotPredictRandomForest", "spotPredictForrester", "spotPredictMlegp", see SPOT
- skipLastSteps: [0] skip the skipLastSteps sequential steps from the selected RES data frame.
- skipIncomplete: [FALSE] if this checkbox is checked (i.e. skipIncomplete==TRUE), then all CONFIGs having *less* than seq.design.maxRepeats repeats are skipped
- nSkip: [0] skip the nSkip worst CONFIGs
- y10Exp: [0] multiply in the RES data frame the column Y (objective function) by the factor $10^{y10Exp}$. This affects the coloring of the metamodel in case "spotReport3d": A surface in the Y-range [1e-3, 1e-2] would have only a single color, but a surface in the range [10,100] will have a richer color scheme.

Usage

tdmPlotResMeta(envT)

Arguments

envT

environment with results as returned from tdmBigLoop or as loaded from an appropriate .RData file. We use here especially envT$resGrid, envT$tdm and envT$tunerVal (the spotConfig from the last tuning experiment).

Details

In case of "spotReport3d", the plot shows the metamodel as colored surface and the true observations of the objective function as black points. Each point stands for a CONFIG, i.e. it is an aggregation (usually: mean) of all repeats for a design point configuration. The options nskip and skipIncomplete allow to suppress certain CONFIGs which may be outliers and thus hinder the view on the 'interesting' region. Note however, that this suppression also affects the metamodel fit, so use these options with care.

In case of "spotReportContour", the plot shows a contour plot of the metamodel with a black point indicating the best CONFIG found.

If envT$tunerVal$meta.compare is TRUE, the quality of the metamodel is evaluated with the following procedure: A RES data frame newRes with N new design CONFIGs (not used during metamodel fit) is taken and their true target value (evaluation of target function) is compared with
the metamodel's value. The mean absolute deviation
\[
\text{sum}(|\text{metamodel}_i - \text{newCONFIG}_i|)/N
\]
is printed and returned in `envT$tunerVal$meta.newMAD`.
The RES data frame `newRes` is either taken from `envT$tunerVal$meta.newRes` (if this element
is not NULL) or it is taken from another member of `envT$resGrid` (preferably from a 'lhd' tuner
result, because it is evenly distributed).
If envT$tunerVal$meta.compare is NULL, it is set to TRUE.

Note

Side Effects: In case of "spotReport3d", one or several RGL plot windows are created which can
be manipulated interactively. A certain RGL window n can be selected with rgl.set(n). An
interactively manipulated RGL window can be saved with rgl.snapshot("myfile.png").

In case of "spotPredictMlegp" there is a new element sC$seq.mlegp.min.nugget=1 set in the source
code which causes the MLEGP fit to become much smoother in the presence of a noisy target
function (nugget effect).

Author(s)

Wolfgang Konen

See Also

dmbigloop

Examples

```r
## Not run:
##
## Read previous tuning results 'envT' from demo02sonar/demoSonar.RData
## (relative to the TDMR package directory).
## Then, tdmPlotResMeta lets you explore interactively the RES data frame(s):
load(paste(find.package("TDMR"), "demo02sonar","demoSonar.RData",sep="/"));
tdmPlotResMeta(envT);

## End(Not run)
```

---

**tdmPreAddMonomials**

*Add monomials of degree 2 to a data frame.*

**Description**

Given the data frame `dset` and a data frame `rx` with the same number of rows, add monomials of
degree 2 to `dset` for all quadratic combinations of the first `PRE.npc` columns of `rx`. The naming of
these new columns is "R1x2" for the combination of cols 1 and 2 and so on (if prefix="R").
Usage
tdmPreAddMonomials(dset, rx, PRE.npc, opts, degree = 2,
    prefix = "R")

Arguments
dset the target data frame
rx a data frame where to draw the monomials from
PRE.npc the number of columns from rx to use (clipped to ncol(rx) if necessary)
opts a list from which we need here the following entries:
    • filename
    • VERBOSE
degree [2] (currently only 2 is supported)
prefix ["R"] character prefix for the monomial column names

Value
data frame dset with the new monomial columns appended. If PRE.npc==0, the data frame is returned unchanged.

Note
CAVEAT: The double for-loop costs some time (e.g. 2-4 sec for ncol(rx)=8 or 10) How to fix: make a version w/o for-loop and w/o frequent assigns to dset (**TODO**)

---------------------------------
tdmPreFindConstVar Find constant columns.
---------------------------------

Description
Find all those columns in data frame dset which are completely constant or completely NA and return a vector with their names.

Usage
tdmPreFindConstVar(dset)

Arguments
dset data frame

Value
name vector of constant columns
**tdmPreGroupLevels**

*Group the levels of factor variable in dset[,colname].*

**Description**

This function reduces the number of levels for factor variables with too many levels. It counts the cases in each level and orders them decreasingly. It binds the least frequent levels together in a new level "OTHER" such that the remaining untouched levels have more than opts$PRE.Xpgroup percent of all cases. OR it binds the levels with least cases together in "OTHER" such that the total number of new levels is opts$PRE.MaxLevel. From these two choices for "OTHER" take the one which binds more variables in column "OTHER".

**Usage**

```r
tdmPreGroupLevels(dset, colname, opts)
```

**Arguments**

- `dset` data frame
- `colname` name of column to be re-grouped
- `opts` list, here we need
  - PRE.Xpgroup [0.99]
  - PRE.MaxLevel [32] (32 is the maximum number of levels allowed for `randomForest`)

**Value**

A data frame with dset[,colname] re-grouped

---

**tdmPreLevel2Target**

*Relate levels of a column with a target (column).*

**Description**

Print for each level of factor variable f which ratio 0 / 1 of the binary target variable it contains and how many cases are in each level

**Usage**

```r
tdmPreLevel2Target(dset, target, f, opts)
```
Arguments

- `dset`: data frame
- `target`: name of target column
- `f`: number of column with factor variable
- `opts`: list, here we need:
  - `opts$thresh_pR`
  - `opts$verbose`

Note

SIDE EFFECTS: some printed output

---

**tdmPreNARoughfix**  
*Replace <NA> values with suitable non <NA> values*

Description

This function replaces <NA> values in a list entry or data frame column with the median (for numeric columns) or the most frequent mode (for factor columns). It does the same as `na.roughfix` in package `randomForest`, but does so faster.

Usage

`tdmPreNARoughfix(object, ...)`

Arguments

- `object`: list or data frame
- `...`: additional arguments

Value

`object`, the list or data frame with <NA> values replaced
**tdmPrePCA.apply**

Apply PCA (Principal Component Analysis) to new data.

---

**Description**

The PCA rotation is taken from `pcaList`, a value returned from a prior call to `tdmPrePCA.train`.

**Usage**

```
tdmPrePCA.apply(dset, pcaList, opts, dtrain = NULL)
```

**Arguments**

- `dset` the data frame with the new data
- `pcaList` a value returned from a prior call to `tdmPrePCA.train`
- `opts` a list from which we need here the following entries:
  - `PRE.knum`: if >0 and if `PRE.PCA`="kernel", take only a subset of `PRE.knum` records from `dset`
  - `PRE.PCA.npc`: if >0, then add for the first `PRE.PCA.npc` PCs the monomials of degree 2 (see `tdmPreAddMonomials`)
  - `PRE.PCA.numericV` vector with all column names in `dset` for which PCA is performed. These columns may contain *numeric* values only.
- `dtrain` [NULL] optional, only needed in case that `dset` is a 0-row-data frame: then we 'borrow' the columns from `dtrain`, the data set returned from `tdmPrePCA.train` in `pca$dsset`.

**Value**

`pca`, a list with entries:

- `dset` the input data frame `dset` with columns `numeric.variables` replaced by the PCs with names `PC1`, `PC2`, ... (in case `PRE.PCA`="linear") or with names `KP1`, `KP2`, ... (in case `PRE.PCA`="kernel") and optional with monomial columns added, if `PRE.PCA.npc>0`
- `numeric.variables` the new column names for PCs and for the monomials

**Author(s)**

Wolfgang Konen, FHK, Mar'2011 - Jan'2012

**See Also**

`tdmPrePCA.train`
PCA (Principal Component Analysis) for numeric columns in a data frame.

Description

tdmPrePCA.train is capable of linear PCA, based on prcomp (which uses SVD), and of kernel PCA (either KPCA, KHA or KFA).

Usage

tdmPrePCA.train(dset, opts)

Arguments

dset the data frame with training (and test) data.

Opts a list from which we need here the following entries:

- PRE.PCA: ["linear" | "kernel" | "none"]
- PRE.knum: if >0 and if PRE.PCA="kernel", take only a subset of PRE.knum records from dset
- PRE.PCA.REPLACE: [T] =T: replace the original numerical columns with the PCA columns; =F: add the PCA columns
- PRE.PCA.npc: if >0, then add for the first PRE.PCA.npc PCs the monomials of degree 2 (see tdmPreAddMonomials)
- PRE.PCA.numericV vector with all column names in dset for which PCA is performed. These columns may contain *numeric* values only.

Value

pca, a list with entries:

- dset the input data frame dset with columns numeric.variables replaced or extended (depending on opts$PRE.PCA.REPLACE) by the PCs with names PC1, PC2, ... (in case PRE.PCA="linear") or with names KP1, KP2, ... (in case PRE.PCA="kernel") and optional with monomial columns added, if PRE.PCA.npc>0. The number of PCs is min(nrows(dset),length(numeric.variables)).
- numeric.variables the new numeric column names (PCs, monomials, and optionally old numericV, if opts$PRE.PCA.REPLACE==F)
- pcaList a list with the items sdev, rotation, center, scale, x as returned from prcomp plus eigval, the eigenvalues for the PCs

Note

CAUTION: Kernel PCA (opts$PRE.PCA="kernel") is currently disabled in code, it *crashes* for large number of records or large number of columns.
Apply SFA (Slow Feature Analysis) to new data.

**Description**

The SFA projection is taken from `sfalist`, a value returned from a prior call to `tdmpresfa.train`.

**Usage**

`tmdpresfa.apply(dset, sfalist, opts, dtrain = NULL)`

**Arguments**

- `dset` the data frame with the new data
- `sfalist` a value returned from a prior call to `tdmpresfa.train`
- `opts` a list from which we need here the following entries:
  - `PRE.SFA.REPLACE`: [T] =T: replace the original numerical columns with the SFA columns; =F: add the SFA columns
  - `PRE.SFA.npc`: if >0, then add for the first `PRE.SFA.npc` PCs the monomials of degree 2 (see `tdmPreAddMonomials`)
  - `PRE.SFA.ODIM`: [5] number of SFA output dimensions (slowest signals) to return
  - `PRE.SFA.numericV` vector with all column names in `dset` for which SFA is performed. These columns may contain *numeric* values only.
- `dtrain` [NULL] optional, only needed in case that `dset` is a 0-row-data frame: then we ‘borrow’ the columns from `dtrain`, the data set returned from `tdmpresfa.train` in `sfa$dset`.

**Value**

`sfa`, a list with entries:

- `dset` the input data frame `dset` with columns `numeric.variables` replaced by the PCs with names `PC1`, `PC2`, ... (in case `PRE.SFA=="linear"`) or with names `KP1`, `KP2`, ... (in case `PRE.SFA=="kernel"`) and optional with monomial columns added, if `PRE.SFA.npc>0`
- `numeric.variables` the new column names for PCs and for the monomials
Author(s)

Wolfgang Konen, Martin Zaefllrer, FHK, Jan’2012 - Feb’2012

See Also

tdmPreSFA.train

tdmPreSFA.train SFA (Slow Feature Analysis) for numeric columns in a data frame.

Description

tdmPreSFA.train uses package rsfa. It is assumed that classification for the variable contained in column response.var is done. SFA seeks features in an expanded function space for which the intra-class variation w.r.t. response.var is as low as possible.

Usage

tdmPreSFA.train(dset, response.var, opts)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dset</td>
<td>the data frame with training (and test) data.</td>
</tr>
<tr>
<td>response.var</td>
<td>the response variable for classification.</td>
</tr>
<tr>
<td>opts</td>
<td>a list from which we need here the following entries:</td>
</tr>
<tr>
<td></td>
<td>• PRE.SFA: [ &quot;linear&quot;</td>
</tr>
<tr>
<td></td>
<td>• PRE.SFA.REPLACE: [T] =T: replace the original numerical columns with</td>
</tr>
<tr>
<td></td>
<td>the SFA columns; =F: add the SFA columns</td>
</tr>
<tr>
<td></td>
<td>• PRE.SFA.npc: if &gt;0, then add for the first PRE.SFA.npc PCs the</td>
</tr>
<tr>
<td></td>
<td>monomials of degree 2 (see tdmPreAddMonomials)</td>
</tr>
<tr>
<td></td>
<td>• PRE.SFA.PPRANGE: [11] number of inputs after preprocessing, they enter</td>
</tr>
<tr>
<td></td>
<td>into expansion</td>
</tr>
<tr>
<td></td>
<td>• PRE.SFA.ODIM: [5] number of SFA output dimensions (slowest signals) to</td>
</tr>
<tr>
<td></td>
<td>return</td>
</tr>
<tr>
<td></td>
<td>• PRE.SFA.numericV vector with all column names in dset which are input</td>
</tr>
<tr>
<td></td>
<td>for SFA. These columns may contain <em>numeric</em> values only.</td>
</tr>
</tbody>
</table>

Value

sfa, a list with entries:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dset</td>
<td>the input data frame dset with columns numeric.variables replaced or</td>
</tr>
<tr>
<td></td>
<td>extended (depending on opts$PRE.SFA.REPLACE) by the SFA components with</td>
</tr>
<tr>
<td></td>
<td>names SF1, SF2, ... and with optional monomial columns added, if PRE.SFA.npc&gt;0</td>
</tr>
</tbody>
</table>
**numeric.variables**  
the new numeric column names of dset, i.e. SFA components, monomials (and optionally PRE.SFA.numericV, if opts$PRE.SFA.REPLACE==F)

**sfaList**  
a list with the items opts (sfa0opts), matrices DSF and SF and many others, as returned from sfaStep

**Author(s)**
Wolfgang Konen, Martin Zaefferer, FHK, Jan’2012 - Feb’2012

**See Also**

tdmPresfaNapply

---

**tdmRandomSeed**  
*Generates pseudo-random random number seeds.*

**Description**
To use this mechanism, create first an object tdmRandomSeed with a call to makeTdmRandomSeed.

**Usage**

```r
tdmRandomSeed()
```

**Value**

In each call to this function a different integer in 0...100001+nCall is returned. This is true even if it is called many times within the same second (where Sys.time() will return the same integer). nCall is the number of calls to this function object.

**Author(s)**
Wolfgang Konen, Patrick Koch <wolfgang.konen@fh-koeln.de>

**See Also**

makeTdmRandomSeed

**Examples**
```r
tdmRandomSeed = makeTdmRandomSeed();
for (i in 1:10) print(c(as.integer(Sys.time()), tdmRandomSeed()));
```
**tdmReadCmd**

*Template function, the default for opts$READ.CMD.*

---

**Description**

This template function is used by **tdmReadData** for data reading.

**Usage**

```r
tdmReadCmd(filename, opts)
```

**Arguments**

- **filename**: this is either opts$filename or opts$filetest
- **opts**: list of options, we need here
  - `dirNtxt`: string, the directory where to read filename
  - `READ.NROW`: [-1] read only that many rows from opts$filename. -1 for 'read all rows'.

**Value**

`dset`, a data frame with all data read

**Note**

This default template function will issue the command

```
 dset=read.csv(file=paste(opts$dir.txt, filename, sep=""),nrow=opts$READ.NROW)
```

which includes the default settings `header=T`, `sep=""` and `dec="."`. The user may alternatively specify his/her own function, the only rules are

- that it has to return a data frame (`dset` or `tset`)
- that the string `opts$READ.CMD` contains an argument `filename`

(`tdmReadData` will call this function with either `opts$filename` or `opts$filetest` replacing `filename`)
Read the data according to the settings in opts.

**Description**

Side effect: Data are saved on .Rdata file if opts$READ.TXT=TRUE (for quicker next-time access).
Next-time access: Data are re-loaded from .Rdata if opts$READ.TXT=FALSE.

**Usage**

```r
tdmReadData(opts)
```

**Arguments**

- **opts**
  - list of options, we need here
    - `filename`: string of filename in opts$dir.txt
    - `filetest`: string of filename test data in opts$dir.txt (only if READ.TST=T)
    - `READ.TXT`: [T] if =T: read from opts$filename, =F: load from .Rdata file
    - `READ.CMD`: ["tdmReadCmd(filename,opts)"] string with a file-read-command with placeholder 'filename'
    - `READ.TST`: [F] if =T: read also test data from filetest
    - `TST.COL`: ["TST.COL"] string, create a column with the name of this string in dset, which has 0 for training and 1 for test data
    - `READ.NROW`: [-1] read only that many rows from opts$filename. -1 for 'read all rows'.

**Details**

When opts$READ.TST==T, the following things happen in `tdmReadData`: Data are read from opts$filename and from opts$filetest. Both data sets are bound together, with a new column opts$TST.COL having '0' for the data from opts$filename and having '1' for the data from opts$filetest. The branch using opts$TST.COL is invoked either with umode="TST" in unbiasedRun or with opts$TST.kind="col" in tdmModCreateCVindex.

**Value**

- dset, a data frame with all data read

**Note**

If opts$READ.TXT=T and opts$READ.NROW>0, then only the first READ.NROW records are read from opts$filename and from opts$filetest. Only the records read are saved to the .Rdata files. This means: when starting tdmReadData again with opts$READ.TXT=F, not more than these READ.NROW records can be retrieved from the relevant .Rdata files. To have again the full data sets on the .Rdata files, you need to invoke tdmReadData at least once with opts$READ.TXT=T and opts$READ.NROW==-1.
tdmRegress

Core regression function of TDMR.

Description

tdmRegress is called by tdmRegressLoop and returns an object of class tdmRegress. It trains a model on training set \(d_{\text{train}}\) and evaluates it on test set \(d_{\text{test}}\). If this function is used for tuning, the test set \(d_{\text{test}}\) plays the role of a validation set.

Usage

```r
tdmRegress(d_train, d_test, d_preproc, 
            response.variables, input.variables, opts)
```

Arguments

- **d_train**: training set
- **d_test**: test set, same columns as training set
- **d_preproc**: data used for preprocessing. May be NULL, if no preprocessing is done (opts$PRE.SFA=="none" and opts$PRE.PCA=="none"). If preprocessing is done, then \(d_{\text{preproc}}\) is usually all non-validation data.
- **response.variables**: name of column which carries the target variable - or - vector of names specifying multiple target columns (these columns are not used during prediction, only for evaluation)
- **input.variables**: vector with names of input columns
- **opts**: additional parameters [defaults in brackets]
  - **SRF.\***: several parameters for sorted_rf_importance (see tdmModelingUtils.r)
  - **RF.\***: several parameters for RF (Random Forest, defaults are set, if omitted)
  - **SVM.\***: several parameters for SVM (Support Vector Machines, defaults are set, if omitted)
  - **filename**: data title
  - **mod.method** ["RF"] the main training method ["RF"|"SVM"|"LM"]: use [Random forest| SVM| linear model] for the main model
  - **mod.SEED**: set the RNG to system time as seed (different RF trainings) =any value: set the random number seed to this value (+i) to get reproducible random numbers. In this way, the model training part (RF, NNET, ...) gets always a fixed seed. (see also TST.SEED in tdmRegressLoop)
  - **OUTTRAFO** [NULL] string, apply a transformation to the output variable

If, on the other hand, the full data sets are stored in the .Rdata files, then opts$READ.TXT=F and opts$READ.NROW=<any value> allows to select a smaller subset of records from the .Rdata files.
**tdmRegress**

fct.postproc [NULL] name of a user-def’d function for postprocessing of predicted output

gr.log =FALSE (def): make scatter plot as-is, =TRUE: transform output x with log(x+1) (x should be nonnegative)

GD.DEVICE if !="non", then make a pairs-plot of the 5 most important variables and make a true-false bar plot

**Value**

res, an object of class tdmRegre, this is a list containing

d_train training set + predicted class column(s)
d_test test set + predicted target output

allRMAE data frame with columns = (rmae.train, rmae.test, theil.train, theil.test, ...) and rows = response variables. Here Theil’s U is based on RMAE (relative mean absolute error).

allRMSE data frame with columns = (rmse.train, rmse.test, theil.train, theil.test, ...) and rows = response variables. Here Theil’s U is based on RMSE (root mean square error).

lastModel the last model built (e.g. the last Random Forest in the case of MOD.method=="RF")

opts parameter list from input, some default values might have been added

The item lastModel is specific for the *last* model (the one built for the last response variable in the last run and last fold)

**Author(s)**

Wolfgang Konen, FHK, Sep’2009 - Jun’2012

**See Also**

print.tdmRegre tdmRegressLoop tdmClassifyLoop

**Examples**

```r
### This example shows a simple data mining process (phase 1 of TDMR) for regression on
### dataset iris.
### The data mining process in tdmRegress calls randomForest as the prediction model.
### It is called for 2 response variables. Therefore, the data frames allRMAE and allRMSE
### have 2 rows.
###
### opts=tdmOptsDefaultsSet()  # set all defaults for data mining process
gdObj <- tdmGraAndLogInitialize(opts);  # init graphics and log file
data(iris)
response.variables=c("Petal.Length","Petal.Width")  # names, not data (!)
input.variables=setdiff(names(iris),response.variables)
opts$rgain.type="rmae"
```
opts$NRUN = 1

idx_train = sample(nrow(iris))[1:110]
d_train = iris[idx_train,]
d_vali = iris[-idx_train,]
res <- tdmRegress(d_train, d_vali, NULL, response.variables, input.variables, opts)

print(res$allRMAE)
print(res$allRMSE)

---

tdmRegressLoop  

Core regression double loop of TDMR returning a TDMregressor object.

Description

tdmRegressLoop contains a double loop (opts$NRUN and CV-folds) and calls tdmRegress. It is called by all R-functions main_.*.

It returns an object of class TDMregressor.

Usage

tdmRegressLoop(dset, response.variables, input.variables, opts)

Arguments

dset  the data frame for which cvi is needed
response.variables  name of column which carries the target variable - or - vector of names specifying multiple target columns (these columns are not used during prediction, only for training and for evaluating the predicted result)
input.variables  vector with names of input columns
opts  a list from which we need here the following entries
NRUN  number of runs (outer loop)
TST.SEED = NULL: leave the random number seed as it is. =any value: set the random number seed to this value to get reproducible random numbers and thus reproducible training-test-set-selection. (only relevant in case TST.kind=="cv" or "rand") (see also MOD.SEED in tdmClassify)
TST.kind  how to create cvi, handed over to tdmModCreateCVindex. If TST.kind="col", then cvi is taken from dset[,opts$TST.col].
GD.RESTART [TRUE] =TRUE/FALSE: do/don’t restart graphic devices
GRAPHDEV ["non"] other ]
Value

result, an object of class `TDMregressor`, this is a list with results, containing

- `opts` the `res$opts` from `tdmRegress`
- `lastRes` last run, last fold: result from `tdmRegress`
- `R_train` RMAE / RMSE on training set (vector of length `NRUN`), depending on `opts$rgain.type=="rmae"` or "rmse"
- `S_train` RMSE on training set (vector of length `NRUN`)
- `T_train` Theil’s U for RMAE on training set (vector of length `NRUN`)
- `_test` — similar, with test set instead of training set —
- `Err` a data frame with as many rows as `opts$NRUN` and columns = (rmae.trn, rmse.trn made.trn, rmae.theil.trn, ntrn, rmae.tst, rmse.tst, made.tst, rmae.theil.tst, ntst)
- `predictions` last run: data frame with dimensions [nrow(dset),length(response.variable)]. In case of CV, all validation set predictions (for each record in dset), in other cases mixed validation / train set predictions.

Author(s)

Wolfgang Konen (<wolfgang.konen@fh-koeln.de>), FHK, Sep’2010 - Jun’2012

See Also

`tdmRegress`, `tdmClassifyLoop`, `tdmClassify`

Examples

```r
### -------- demo/demo00-1regress.r --------
### This demo shows a simple data mining process (phase 1 of TDMR) for regression on
### dataset iris.
### The data mining process in tdmRegressLoop calls randomForest as the prediction model.
### It is called `opts$NRUN` times with different random train-validation set splits.
### Therefore data frame `result$Err` has 2 rows.
###
### opts=tdmOptDefaultsSet()                # set all defaults for data mining process
gObj <- tdmGraAndLogInitialize(opts);    # init graphics and log file
data(iris)
response.variables="Petal.Length"       # names, not data (!)
input.variables=setdiff(names(iris),"Petal.Length")
opts$rgain.type="rmae"

result = tdmRegressLoop(iris,response.variables,input.variables,opts)
print(result$Err)
```

---

`tdmRegressLoop` 57
tdmRegresSummary

Print summary output for result from tdmRegressLoop and add result$y.

Description

result$y is "OOB RMAE" on training set for methods RF or MC.RF. result$y is "RMAE" on test set (=validation set) for all other methods. result$y is the quantity which the tuner seeks to minimize.

Usage

tdmRegresSummary(result, opts, dset = NULL)

Arguments

result return value from a prior call to tdmRegressLoop, an object of class TDMregressor.

opts a list from which we need here the following entries

NRUN number of runs (outer loop)
method
VERBOSE
dset [NULL] if !=NULL, attach it to result

Value

result, an object of class TDMregressor, with result$y, result$sd.y (and optionally also result$dset) added

Author(s)

Wolfgang Konen, FHK, Sep’2010 - Oct’2011

See Also
tdmRegress, tdmRegressLoop, tdmClassifySummary
**tdmROCR.TDMclassifier**  
*Interactive plot of ROC, lift or other charts for a TDMclassifier object.*

**Description**

Brings up a *twiddle* user interface, where the user may select a part of the dataset ("training" or "validation"), a run number (if `Opts(x)$NRUN>1`) and a type-of-chart, see `tdmROCRbase` for details. Using `tdmROCRbase`, the appropriate chart is plotted on the current graphics device.

**Usage**

```r
## S3 method for class 'TDMclassifier'
tdmROCR(x, ...)
```

**Arguments**

- `x`  
  return value from a prior call to `tdmClassifyLoop`, an object of class `TDMclassifier`.

- `...`  
  – currently not used –

**Value**

The area under the curve plotted most recently.

**Note**

Side effect: Ror each chart, calculate and print the area between the curve and the bottom line (\(y=1.0\) for typ=="lift", \(y=0.0\) else).

**See Also**

- `tdmClassifyLoop`  
- `tdmROCRbase`

**Examples**

```r
## Not run:
setwd(paste(find.package("TDMP"), "demo82sonar",sep="/"));
source("main_sonar.r");
result = main_sonar();
tdmROCR(result);

## End(Not run)
```
**tdmROCRbase**

*Single plot of ROC, lift or other chart for a TDMclassifier object.*

**Description**

Single plot of ROC, lift or other chart for a TDMclassifier object.

**Usage**

```r
dtmROCRbase(x, dataset = "validation", nRun = 1,
             typ = "ROC", noPlot = FALSE, ...)
```

**Arguments**

- `x`: return value from a prior call to `tdmClassifyLoop`, an object of class TDMclassifier.
- `dataset`: ["validation"] which part of the data to use, either "training" or "validation".
- `nRun`: [1] if `x` contains multiple runs, which run to show (1,...,opts(x)$NRUN).
- `typ`: ["ROC"] which chart type, one out of ("ROC","lift","precRec") for (ROC, lift, precision-recall)-chart (see `performance` in package ROCR for more details):
  - "ROC": receiver operating curve, TPR vs. FPR, with TPR=TP/(TP+FN)=TP/P and FPR=FP/(FP+TN)=FP/N (true and false positive rate).
  - "lift": lift chart, LIFT vs. RPP, with LIFT=TPR/RPR with random positive rate RPR=P/(P+N) and RPP=(TP+FP)/(P+N) (rate of pos. predictions).
  - "precRec": precision-recall-chart, PREC vs. RECALL, with PREC=TP/(TP+FP) and RECALL=TP/P (same as TPR).
- `noPlot`: [FALSE] if TRUE, suppress the plot, return only the area under curve.
  - currently not used

**Value**

The area between the curve and the bottom line y=0.0 in the case of `typ="ROC" | typ="precRec"` or the area between the curve and the bottom line y=1.0 in the case of `typ="lift"`.  
If object `x` does not contain a prediction score, a warning is issued and the return value is NULL.

**See Also**

- `tdmClassifyLoop`  
- `tdmROCR.TDMclassifier`

**Examples**

```r
### Run task SONAR with "area under ROC curve" as performance measure (rgain.type="arROC").
### Other settings are similar to demo01-1sonar.r (level 1 of TDMR).
### Finally, plot ROC curve for validation data set and
### plot lift chart for training data set
###```
**tdmSplitTestData**  
*Read and split the task data and return a TDMdata object.*

### Description

Read the task data using `tdmReadData` and split them into a test part and a training/validation-part.

### Usage

```r
tdmSplitTestdata(opts, tdm, nExp = 0, dset = NULL)
```

### Arguments

- **opts**  
a list from which we need here the elements
  - `READ.INI`: [T] =T: do read and split, =F: return NULL
  - `READ.*`: other settings for `tdmReadData`
  - `filename`: needed for `tdmReadData`
  - `filetest`: needed for `tdmReadData`
  - `TST.testFrac`: [0,1] set this fraction of the data aside for testing
  - `TST.COL`: string with name for the partitioning column, if `tdm$umode` is not "SP_T". (If `tdm$umode`="SP_T", then `TST.COL="tdmSplit"` is used.)

- **tdm**  
a list from which we need here the elements
  - `mainFile`: if not NULL, set working dir to `dir(mainFile)` before executing `tdmReadData`
  - `umode`: ["RSUB"|"CV"|"TST"|"SP_T"], how to divide in training/validation data for tuning and test data for the unbiased runs
tdmStartSpot

- SPLIT.SEED: if not NULL, set random number generator to this value --> deterministic test set split
- stratified: [NULL] if not NULL, do the split by stratified sampling (at least one record of each class level found in dset[,tdm$stratified] shall appear in the train-vali-set, recommended for classification)

nExp [0] experiment counter, used to select a reproducible different seed, if tdm$SPLIT.SEED!=NULL
dset [NULL] if non-NULL, reading of dset is skipped and the given data frame dset is used.

Value
dataObj, either NULL (if opts$READ.INI==FALSE) or an object of class TDMdata containing dset a data frame with the complete data set
TST.COL string, the name of the column in dset which has a 1 for records belonging to the test set and a 0 for train/vali records. If tdm$umode=="SP_T", then TST.COL="tdmSplit", else TST.COL=opts$TST.COL.
filename opts$filename, from where the data were read

Use the accessor functions dsetTrnVa.TDMdata and dsetTest.TDMdata to extract the train/vali and the test data, resp., from dataObj.

Known caller: tdmBigLoop

Author(s)
Wolfgang Konen (<wolfgang.konen@fh-koeln.de>), 2012 - 2013

See Also
dsetTrnVa.TDMdata, dsetTest.TDMdata, tdmReadData, tdmBigLoop

dtmStartSpot Function called by spot to evaluate a DM task during a SPOT tuning run.

Description
This function is called by spot for evaluations during a tuning run. It accumulates in spotConfig$alg.currentResult the RES data frame of all evaluations and in spotConfig$alg.currentBest the BST data frame of the so far best solution.

Usage
tdmStartSpot(spotConfig)
Arguments

spotConfig

the list of configurations for SPOT. Besides the usual SPOT settings, this list has
to contain an element tdm with the mandatory elements

  • tdm$mainFunc: the function name of the DM task to execute (e.g. "main_sonar")
    It is expected that mainFunc returns a list result with the element result$y,
    the quantity to be minimized by SPOT or other tuners.

Optional, tdm may define

  • tdm$mainFile: e.g. "myDir/main_sonar.r", the R file of the DM task to
    source and dirname(tdm$mainFile) is the directory where tdm$mainFunc
    is executed.

Details

spot is invoked from spotTuner or lhdtuner. The latter is simply a SPOT-run with all budget
devoted to the initial design.

Value

spotConfig

Author(s)

Wolfgang Konen

See Also

spot, spotTuner, lhdtuner, tdmDispatchTuner

Description

Read the best solution of a parameter-tuning run from envT$bst, execute with these best parameters
the function tdm$mainFunc (usually a classification or regression machine learning task), to see
whether the result quality is reproducible on independent test data or on independently trained
models.

Usage

unbiasedRun(confFile, envT, dataObj = NULL,
    finals = NULL, umode = "RSUB", withParams = FALSE,
    tdm = NULL)
unbiasedRun

**Arguments**

- `confFile`: the .conf filename, e.g. "appAcid_02.conf"
- `envT`: environment, from which we need the objects
- `bst`: data frame containing best results (merged over repeats)
- `res`: data frame containing all results
- `theTuner`: ["spot"] string
- `spotConfig`: [NULL] a list with SPOT settings. If NULL, try to read spotConfig from confFile.
- `dataObj`: [NULL] contains the pre-fetched data from which we use here the test-set part. If NULL, set it to tdmSplitTestData(opts, tdm)
- `finals`: [NULL] a one-row data frame to which new columns with final results are added
- `umode`: ["RSUB" (default) | "CV" | "TST" | "SP_T" ], how to divide in training and test data for the unbiased runs:
  - "RSUB" random subsampling into (1-tdm$TST.testFrac)% training and tdm$TST.testFrac% test data
  - "CV" cross validation (CV) with tdm$nrun folds
  - "TST" all data in opts$filename are used for training, all data in opts$filetest for testing
  - "SP_T" 'split_test': prior to tuning, the data set was split by random subsampling into tdm$TST.testFrac% test and (1-tdm$TST.testFrac)% training-vali data, tagged via column "tdmSplit". Tuning was done on training-vali data. Now we use column "tdmSplit" to select the test data for unbiased evaluation. Training during unbiased evaluation is done on a fraction tdm$TST.trnFrac of the training-vali data
- `withParams`: [FALSE] if =TRUE, add columns with best parameters to data frame `finals` (should be FALSE, if different runs have different parameters)
- `tdm`: a list with TDM settings from which we use here the elements
  - `mainFunc`: the function to be called for unbiased evaluations
  - `mainFile`: change to the directory of mainFile before starting mainFunc
  - `nrun`: [5] how often to call the unbiased evaluation
  - `nfold`: [10] how many folds in CV (only relevant for umode="CV")
  - `TST.testFrac`: [0.2] test set fraction (only relevant for umode="RSUB" or="SP_T")

The defaults in ' [...] ' are set by `tdmDefaultsFill`, if they are not defined on input.

**Value**

- `finals`: a one-row data frame with final results

**Note**

Side Effects: The list result, an object of class TDMclassifier or TDMregressor as returned from tdm$mainFunc is written onto envT$result.
If envT$spotConfig is NULL, it is constructed from confFile.
If `spotConfig$opts` (list with all parameter settings for the DM task) is NULL, we try to read it from `spotConfig$io.apdFileName`. This will issue a warning ‘... might not work in parallel mode’, but is perfectly fine for non-parallel mode.

**Author(s)**

Wolfgang Konen, FHK, 2010 - 2013

**See Also**

`tdmBigLoop`, `TDClassifier`, `TDMRegressor`

**Examples**

```r
## Load the best results obtained in a prior tuning for the configuration
## "sonar_04.conf" with tuning method "spot". The result envT from a prior
## run of tdmBigLoop with this .conf is read from demo02sonar/demoSonar.RData.
## Run task main_sonar again with these best parameters, using the default
## settings from tdmDefaultsFill: umode="RSUB", tdm$nrun=5 and tdm$TST.testFrac=0.2.
oldwd <- getwd();
## The best results are read from demo02sonar/demoSonar.RData relative to
## the TDMR package directory.
setwd(paste(find.package("TDMR"), "demo02sonar", sep="/"));
load("demoSonar.RData");
source("main_sonar.r");
finals <- unbiasedRun("sonar_04.conf", envT, tdm=envT$tdm);
print(finals);
setwd(oldwd);
```
Index

*Topic **data**
  TDMR-package, 3

*Topic **learning**
  TDMR-package, 3

*Topic **machine**
  TDMR-package, 3

*Topic **mining**
  TDMR-package, 3

*Topic **package**
  TDMR-package, 3

*Topic **tuning**
  TDMR-package, 3

bfgsTuner, 10
cma_jTuner, 10, 20
cmaesTuner, 10
dsetTest.TDMdata, 4, 5, 62
dsetTrnVa.TDMdata, 4, 4, 62

lhdTuner, 10, 63
makeTdmRandomSeed, 51

Opts, 5, 59, 60

parallel, 3
df, 28
performance, 60
png, 29
powell, 10
powellTuner, 10
prcomp, 48
predict.TDMclassifier
  (predict.TDMenvir), 6
predict.TDMenvir, 6
predict.TDMregressor
  (predict.TDMenvir), 6
print, 11
print.tdmClass, 15

print.tdmClass (print.TDMclassifier), 7
print.TDMclassifier, 7, 17, 18
print.TDMdata, 8
print.tdmRegre, 55
print.tdmRegre (print.TDMregressor), 8
print.TDMregressor, 8

randomForest, 45
rCMA, 3, 10
rSFA, 38, 50

sfaBootstrap, 38
sfaStep, 51
SPOT, 3, 42, 62, 63
spot, 10, 62, 63
spotTuner, 10, 63
tdmBigLoop, 3, 9, 19, 20, 23, 25, 26, 42, 43, 62, 65
tdmBindResponse, 13
tdmClass, 5–7
tdmClassifier, 5–7, 11, 16, 17, 59, 60, 64, 65
tdmClassifier (tdmClassifyLoop), 16
tdmClassify, 7, 13, 16–18, 32, 38–41, 56, 57
tdmClassifyLoop, 3, 7, 13–15, 16, 18, 37, 39, 40, 55, 57, 59, 60
tdmClassifySummary, 7, 18, 58
tdmData, 4, 5, 8, 10, 61, 62
tdmData (tdmSplitTestData), 61
tdmDefaultsFill, 3, 9, 19, 23, 25, 40, 64
tdmDispatchTuner, 9, 11, 12, 19, 63
tdmEmbedDataFrame, 21
tdmEnvir, 5, 6, 23, 24
tdmEnvir (tdmEnvTMakeNew), 23
tdmEnvTAddBstRes, 10, 21
tdmEnvTAddGetters, 22, 22
tdmEnvTLoad, 22
tdmEnvTMakeNew, 23, 24, 26
tdmEnvTReadApd, 24
tdmEnvTSensi, 24
tdmExecSpotStep, 25
tdmGraAndLogFinalize, 26
tdmGraAndLogInitialize, 27
tdmGraphicCloseDev, 27
tdmGraphicCloseWin, 28
tdmGraphicInit, 26–28, 28, 29
tdmGraphicNewWin, 29
tdmGraphicToTop, 29
tdmMapDesApply, 30, 30, 31
tdmMapDesLoad, 30, 30
tdmModConfMat, 31, 39
tdmModCreateCVindex, 16, 32, 37, 53, 56
tdmModSortedRFimport, 14, 34, 34, 36, 38
tdmModVote2Target, 35
tdmOptsDefaultsFill, 36, 40
tdmOptsDefaultsSet, 3, 36, 40
tdmParaBootstrap, 40, 41
tdmPlotResMeta, 42
tdmPreAddMonomials, 43
tdmPreFindConstVar, 44
tdmPreGroupLevels, 38, 45
tdmPreLevel2Target, 45
tdmPreNARoughfix, 46
tdmPrePCA.apply, 47, 49
tdmPrePCA.train, 47, 48
tdmPreSFA.apply, 49, 51
tdmPreSFA.train, 49, 50, 50
TD MR (TD MR-package), 3
TD MR-package, 3
tdmRandomSeed, 14, 16, 37, 38, 51
tdmReadCmd, 37, 52
tdmReadData, 37, 52, 53, 53, 61, 62
tdmRegre, 5, 6, 8, 9
tdmRegress, 9, 17, 38–40, 54, 56–58
tdmRegressLoop, 3, 9, 15, 17, 54, 55, 56, 58
TD M regressor, 5, 6, 8, 9, 11, 56, 57, 64, 65
TD M regressor (tdmRegressLoop), 56
tdmRegressSummary, 9, 18, 58
tdmROCR. TDMclassifier, 59, 60
tdmROCRbase, 32, 59, 60
tdmSplitTestData, 4, 5, 8, 10, 20, 61
tdmStartSpot, 19, 62
twiddle, 42, 59
unbiasedRun, 4, 6, 9, 11, 12, 19, 53, 63