

# Package ‘TDMR’

May 1, 2019

**Type** Package

**Title** Tuned Data Mining in R

**Version** 2.1

**Date** 2019-05-01

**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 the complete tuning of a data mining task (predictive analytics, that is classification and regression). Preprocessing parameters and modeling parameters can be tuned simultaneously. It incorporates a variety of tuners (among them 'SPOT' and 'CMA' with package 'rCMA') and allows integration of additional tuners. Noise handling in the data mining optimization process is supported, see Koch et al. (2015) <doi:10.1016/j.asoc.2015.01.005>.

**License** GPL (>= 2)

**Depends** R (>= 3.0.0), SPOT (>= 2.0), twiddler

**Suggests** cmaes, parallel, e1071, ROCR, randomForest, rCMA, rSFA

**Imports** testit, methods, adabag

**Collate** 'defaultSC.R' 'defaultOpts.R' 'makeTdmRandomSeed.r'  
'printTDMclassifier.r' 'printTDMregressor.r' 'tdmBigLoop.r'  
'tdmClassify.r' 'tdmClassifyLoop.r' 'tdmDefaultsFill.r'  
'tdmDispatchTuner.r' 'tdmEnvTMakeNew.r' 'tdmGeneralUtils.r'  
'tdmGraphicUtils.r' 'tdmMapDesign.r' 'tdmMetacostRf.r'  
'tdmModelingUtils.r' 'tdmOptsDefaults.r' 'tdmParaBootstrap.r'  
'tdmPreprocUtils.r' 'tdmReadAndSplit.r' 'tdmReadDataset.r'  
'tdmRegress.r' 'tdmRegressLoop.r' 'tdmROCR.r' 'tdmStartSpot2.r'  
'tdmStartOther.r' 'tdmTuneIt.r' 'unbiasedRun.r'

**RoxygenNote** 6.0.1

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2019-05-01 17:10:03 UTC

**R topics documented:**

TDMR-package . . . . .	3
defaultOpts . . . . .	4
defaultSC . . . . .	5
dsetTest.TDMdata . . . . .	6
dsetTrnVa.TDMdata . . . . .	7
Opts . . . . .	7
predict.TDMenvir . . . . .	8
print.TDMclassifier . . . . .	9
print.TDMdata . . . . .	10
print.TDMregressor . . . . .	11
setParams . . . . .	11
tdmBigLoop . . . . .	12
tdmBindResponse . . . . .	15
tdmClassify . . . . .	16
tdmClassifyLoop . . . . .	18
tdmClassifySummary . . . . .	20
tdmDefaultsFill . . . . .	21
tdmEnvTAddBstRes . . . . .	23
tdmEnvTAddGetters . . . . .	24
tdmEnvTGetOpts . . . . .	24
tdmEnvTLoad . . . . .	25
tdmEnvTMakeNew . . . . .	25
tdmEnvTReport . . . . .	26
tdmEnvTReportSens . . . . .	27
tdmEnvTSetOpts . . . . .	27
tdmEnvTUpdate . . . . .	28
tdmGraAndLogFinalize . . . . .	28
tdmGraAndLogInitialize . . . . .	29
tdmGraphicCloseDev . . . . .	29
tdmGraphicCloseWin . . . . .	30
tdmGraphicInit . . . . .	30
tdmGraphicNewWin . . . . .	31
tdmGraphicToTop . . . . .	31
tdmMapDesApply . . . . .	32
tdmMapDesLoad . . . . .	32
tdmModConfmat . . . . .	33
tdmModCreateCVindex . . . . .	35
tdmModSortedRFimport . . . . .	36
tdmModVote2Target . . . . .	37
tdmOptsDefaultsSet . . . . .	38
tdmParaBootstrap . . . . .	43
tdmPreAddMonomials . . . . .	44
tdmPreFindConstVar . . . . .	45
tdmPreGroupLevels . . . . .	45
tdmPreLevel2Target . . . . .	46
tdmPreNAroughfix . . . . .	46

tdmPrePCA.apply . . . . .	47
tdmPrePCA.train . . . . .	48
tdmPreSFA.apply . . . . .	49
tdmPreSFA.train . . . . .	50
tdmRandomSeed . . . . .	51
tdmReadAndSplit . . . . .	52
tdmReadDataset . . . . .	53
tdmReadTaskData . . . . .	54
tdmRegress . . . . .	55
tdmRegressLoop . . . . .	57
tdmRegressSummary . . . . .	59
tdmROCR.TDMclassifier . . . . .	60
tdmROCRbase . . . . .	61
tdmTuneIt . . . . .	62
unbiasedRun . . . . .	65

<b>Index</b>	<b>68</b>
--------------	-----------

---

TDMR-package	<i>Tuned Data Mining in R</i>
--------------	-------------------------------

---

## Description

Tuned Data Mining in R

## Details

Package:	TDMR
Type:	Package
Version:	2.1
Date:	01.05.2019
License:	GPL (>= 2)
LazyLoad:	yes

TDMR 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 parameters and model building parameters simultaneously. Preprocessing often includes feature generation.
- 3) Support for multiple tuning experiments (different settings, repetitions with different resamplings, ...).
- 4) Easy parallelization of those experiments with the help of R package [parallel](#).
- 5) Extensibility: New tuning parameters, new preprocessing tools, model builders and even new tuners can be added easily.

The main entry point functions are [tdmClassifyLoop](#), [tdmRegressLoop](#), [tdmTuneIt](#), and [tdmBigLoop](#).

See [tdmOptsDefaultsSet](#) and [tdmDefaultsFill](#) for an overview of adjustable TDMR-parameters.

### Author(s)

Wolfgang Konen (<[wolfgang.konen@th-koeln.de](mailto:wolfgang.konen@th-koeln.de)>), Patrick Koch

### References

<http://lwibs01.gm.fh-koeln.de/blogs/ciop/research/tuned-data-mining/>

---

defaultOpts

*Default settings for the data mining part of TDMR (list opts).*

---

### Description

Sets suitable defaults for the data mining part of TDMR.

### Usage

```
defaultOpts()
```

### Details

With the call `setParams(myOpts, defaultOpts())` it is possible to extend a partial list `myOpts` to a list containing all `opts`-elements (the missing ones are taken from `defaultOpts()`). If `myOpts` has an element not present in `defaultOpts()`, this element is not taken and a warning is issued.

With `setParams(myOpts, defaultOpts(), keepNotMatching=TRUE)` also elements of `myOpts` not present in `defaultOpts()` are taken (no warnings).

### Value

a list with the elements according to [tdmOptsDefaultsSet](#)

### Author(s)

Wolfgang Konen (<[wolfgang.konen@th-koeln.de](mailto:wolfgang.konen@th-koeln.de)>), Samineh Bagheri, THK, 2018

### See Also

[setParams](#), [defaultSC](#)

---

defaultSC	<i>Default settings for the spotConfig part of TDMR.</i>
-----------	--

---

### Description

Sets suitable defaults for the spotConfig part of TDMR.

### Usage

```
defaultSC()
```

### Details

With the call `setParams(mySC,defaultSC())` it is possible to extend a partial list mySC to a list containing all sC-elements (the missing ones are taken from `defaultSC()`). If mySC has an element not present in `defaultSC()`, this element is not taken and a warning is issued.

With `setParams(mySC,defaultSC(),keepNotMatching=TRUE)` also elements of mySC not present in `defaultSC()` are taken (no warnings).

### Value

a list with the following elements (the values in parantheses [] are the defaults):

alg.roi	[ "NEEDS_TO_BE_SET" ] a data frame with columns lower, upper, type, row.names, each a vector with as many entries as there are parameter to be tuned
opts	[ "NEEDS_TO_BE_SET" ]
sCName	[ "NEEDS_TO_BE_SET.conf" ] a string ending on ".conf", the configuration name
OCBA	[ FALSE ] see <a href="#">spotControl</a>
plot	[ FALSE ] TRUE: make a line plot showing progress
seedSPOT	[ 1 ] see <a href="#">spotControl</a>
funEvals	[ 50 ] the budget, max number of algo evaluations
design	[ designLHD ] function that creates initial design, see <a href="#">spotControl</a>
designControl.size	[ 10 ] number of initial design points (former init.design.size)
designControl.replicates	[ 2 ] number of initial repeats (former init.design.repeats)
replicates	[ 2 ] number of repeats for the same model design point
noise	[ TRUE ] whether the object function has noise or not (Note: TRUE is required if replicates>1 (!))
seq.merge.func	[ mean ] how to merge Y over replicates: mean or min
model	[ buildKriging ] function that builds the surrogate model, see <a href="#">spotControl</a>
optimizer	[ optimLHD ] function that optimizes on surrogate, see <a href="#">spotControl</a>
optimizerControl.funEvals	[ 100 ] optimizer budget (former seq.design.size)
optimizerControl.retries	[ 2 ] optimLHD retries (former seq.design.retries)

**Author(s)**

Wolfgang Konen (<wolfgang.konen@th-koeln.de>) THK, 2018

**See Also**

[setParams](#), [defaultOpts](#)

---

dsetTest.TDMdata      *Return test data of [TDMdata](#) object*

---

**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 [tdmReadAndSplit](#), an object of class [TDMdata](#).  
`...`                may contain `nExp`, experiment number, needed only if `x$tdm$umode=="SP_T"`:  
                      add `nExp` to seed when randomly splitting in train and test data [default: `nExp=0`]

**Value**

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

**Author(s)**

Wolfgang Konen, THK

**See Also**

[unbiasedRun](#) [dsetTrnVa.TDMdata](#) [tdmReadAndSplit](#)

---

dsetTrnVa.TDMdata      *Return train-validation data of [TDMdata](#) object*

---

### Description

Return the train-validation part of a [TDMdata](#) object containing the task data.

### Usage

```
## S3 method for class 'TDMdata'  
dsetTrnVa(x, ...)
```

### Arguments

x                    return value from a prior call to [tdmReadAndSplit](#), an object of class [TDMdata](#).  
...                   may contain nExp, experiment number, needed only if x\$tdm\$umode=="SP\_T":  
                      add nExp to seed when randomly splitting in train and test data [default: nExp=0]

### Value

dset, a data frame with all train-validation records

### Author(s)

Wolfgang Konen, THK

### See Also

[dsetTest.TDMdata](#) [tdmReadAndSplit](#)

---

Opts                    *Return the list 'opts'.*

---

### Description

Returns the list opts from objects of class [TDMenvir](#), [TDMclassifier](#), [TDMregressor](#), [tdmClass](#) or [tdmRegre](#).

**Usage**

```

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, ...)

```

**Arguments**

x            an object of class [TDMenvir](#), [TDMclassifier](#), [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 [TDMenvir](#), [TDMclassifier](#), [TDMregressor](#). The prediction is based on the (last) model trained during [unbiasedRun](#).

**Usage**

```

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

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

```



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

### Arguments

object	an object of class <a href="#">TDMenvir</a> , <a href="#">TDMclassifier</a> , <a href="#">TDMregressor</a> 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

```
## Not run:
## This example requires that demo04cpu.r is executed first (it will write demo04cpu.RData)
path <- paste(find.package("TDMR"), "demo01cpu/", sep="/");
tdm <- list( filenameEnvT="demo04cpu.RData" ); # file with environment envT
load(paste(path,tdm$filenameEnvT,sep="/"));

# take only the first 15 records:
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](#) 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, THK

**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**

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

**Arguments**

x                    return value from a prior call to [tdmReadAndSplit](#), an object of class [TDMdata](#).

...                    currently not used

**Author(s)**

Wolfgang Konen, FHK

**See Also**

[tdmReadAndSplit](#)

---

print.TDMregressor      *Print an overview for a [TDMregressor](#) object.*

---

### Description

Print an overview for a [TDMregressor](#) or [tdmRegre](#) object.

### Usage

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

## 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

### Author(s)

Wolfgang Konen, THK

### See Also

[tdmRegress](#), [tdmRegressSummary](#), [TDMregressor](#)

---

setParams                    *Merge the parameters from a partial list and the default list*

---

### Description

Merge the parameters from a partial list and the default list

### Usage

```
setParams(opts, defaultOpt, keepNotMatching = FALSE)
```

**Arguments**

**opts** a partial list of parameters  
**defaultOpt** a list with default values for every element  
**keepNotMatching** [FALSE] if TRUE, copy the elements appearing in **opts**, but not in **defaultOpt** to the return value. If FALSE, do not copy them, but issue a warning.

**Value**

a list combined from **opts** and **defaultOpt** where every available element in **opts** overrides the default. For the rest of the elements the value from **defaultOpt** is taken.  
 A warning is issued for every element appearing in **opts** but not in **defaultOpt** (only if **keepNotMatching**==FALSE).

**Author(s)**

Wolfgang Konen (<wolfgang.konen@th-koeln.de>), Samineh Bagheri

**See Also**

[defaultSC](#), [defaultOpts](#)

---

 tdmBigLoop

*Tuning and unbiased evaluation in a big loop.*

---

**Description**

For each configuration object **.conf** 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\$nExperim** times. Thus we have for each triplel

(**confName**, **nExp**, **theTuner**)

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

**confName** in **tdm\$runList**  
**nExp** in 1,...,**tdm\$nExperim**  
**theTuner** in **tdm\$tuneMethod**

**Usage**

**tdmBigLoop**(envT, dataObj = NULL)

**Arguments**

envT	an environment containing on input at least the element <code>tdm</code> (a list with general settings for TDMR, see <code>tdmDefaultsFill</code> ), which has at least the elements <code>tdm\$runList</code> vector of configuration names <code>.conf</code>
dataObj	[NULL] optional object of class <code>TDMdata</code> (the same for all runs in big loop). If it is NULL, it will be constructed here with the help of <code>tdmReadAndSplit</code> . Then it can be different for each configuration object in the big loop.

**Details**

`tdm` refers to `envT$tdm`.

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 `cma_es` 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.

**Value**

environment `envT`, containing the results

<code>res</code>	data frame with results from last tuning (one line for each call of <code>tdmStart*</code> )
<code>bst</code>	data frame with the best-so-far results from last tuning (one line collected after each (SPO) step)
<code>resGrid</code>	list with data frames <code>res</code> from all tuning runs. Use <code>envT\$getRes(envT, confFile, nExp, theTuner)</code> to retrieve a specific <code>res</code> .
<code>bstGrid</code>	list with data frames <code>bst</code> from all tuning runs. Use <code>envT\$getBst(envT, confFile, nExp, theTuner)</code> to retrieve a specific <code>bst</code> .
<code>theFinals</code>	data frame with one line for each triple ( <code>confFile, nExp, tuner</code> ), each line contains summary information about the tuning run in the form: <code>confFile tuner nExp [params] NRUN NEVAL RGain.bst RGain.* sdR.*</code> where <code>[params]</code> is written depending on <code>tdm\$withParams</code> . 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 preceding `RGain` or `RMAE`. `RGain.best` 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$umode` (the possible values are explained in [unbiasedRun](#)).

<code>result</code>	object of class <code>TDMclassifier</code> or <code>TDMregressor</code> . This is a list with results from <code>tdm\$mainFunc</code> as called in the last unbiased evaluation using the best parameters found during tuning. Use <code>print(envT\$result)</code> to get more info on such an object of class <code>TDMclassifier</code> .
<code>tunerVal</code>	an object with the return value from the last tuning process. For every tuner, this is the list <code>spotConfig</code> , containing the SPOT settings plus the TDMR settings in elements <code>opts</code> and <code>tdm</code> . Every tuner extends this list by <code>tunerVal\$alg.currentResult</code> and <code>tunerVal\$alg.currentBest</code> , see <a href="#">tdmDispatchTuner</a> . 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: A compressed version of `envT` is saved to file `tdm$filenameEnvT` (default: `<runList[1]>.RData`) in directory `tdm$path`. If `tdm$path==NULL` use the current directory.

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

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@th-koeln.de](mailto:wolfgang.konen@th-koeln.de)), THK, Patrick Koch

### See Also

[tdmDispatchTuner](#), [unbiasedRun](#)

### Examples

```
### 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,
### http://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+%28Sonar,+Mines+vs.+Rocks%29).
### 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 controlSC() (control_sonar.r). 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.

## path is the dir with data and main_*.r file:
path <- paste(find.package("TDMR"), "demo02sonar", sep="/");
#path <- paste("../..inst", "demo02sonar", sep="/");

## control settings for TDMR
tdm <- list( mainFunc="main_sonar"
            , runList = c("sonar_04.conf")
            , umode="CV"           # { "CV" | "RSUB" | "TST" | "SP_T" }
            , tuneMethod = c("lhd")
            , filenameEnvT="exBigLoop.RData" # file to save environment envT
            , nrun=1, nfold=2         # repeats and CV-folds for the unbiased runs
            , nExperim=1
            , optsVerbosity = 0      # the verbosity for the unbiased runs
            );
source(paste(path,"main_sonar.r",sep="/")); # main_sonar, readTrnSonar

### This demo is for example and help (more meaningful, a bit higher budget)
source(paste(path,"control_sonar.r",sep="/")); # controlDM, controlSC

ctrlSC <- controlSC();
ctrlSC$opts <- controlDM();

# construct envT from settings given in tdm & sCList
envT <- tdmEnvTMakeNew(tdm,sCList=list(ctrlSC));
dataObj <- tdmReadTaskData(envT,envT$tdm);
envT <- tdmBigLoop(envT,dataObj=dataObj); # start the big tuning loop

```

---

tdmBindResponse	<i>Bind a column to a data frame.</i>
-----------------	---------------------------------------

---

### 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 bound to data frame d

**Value**

data frame d with column added

---

tdmClassify	<i>Core classification function of TDMR.</i>
-------------	--

---

**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, tsetStr = c("Validation", "validation"))
```

**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 (dif-
  ferent RF trainings).
  =any value: set the random number seed to this value (+i) to get repro-
  ducible 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) (cur-
  rently used only by methods RF, MC.RF and by tdmModSortedRFimport)
fct.postproc [NULL] name of user-def'd function for postprocessing of pre-
  dicted 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
tsetStr [c("Validation", "validation")]

```

### 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

<code>d_train</code>	training set + predicted class column(s)
<code>d_test</code>	test set + predicted class column(s)
<code>d_dis</code>	disregard set + predicted class column(s)
<code>avgEVAL</code>	list with evaluation measures, averaged over all response variables
<code>alleval</code>	data frame with evaluation measures, one row for each response variable
<code>lastCmTrain</code>	a list with evaluation info for training set (confusion matrix, gain, class errors, ...)
<code>lastCmVali</code>	a list with evaluation info for validation set (confusion matrix, gain, class errors, ...)
<code>lastModel</code>	the last model built (i.e. for the last response variable)
<code>lastProbs</code>	a list with three probability matrices (row: records, col: classes) <code>v_train</code> , <code>v_test</code> , <code>v_dis</code> , if the model provides probabilities; NULL else.
<code>lastPred</code>	name of the colum where the prediction of the last model is appended to the datasets <code>d_train</code> , <code>d_test</code> and <code>d_dis</code>
<code>predProb</code>	a list with two data frames <code>Trn</code> and <code>Val</code> . They contain at least a column <code>IND.dset</code> (index of each train / validation record into data frame <code>dset</code> ). 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, THK, 2013

### See Also

[print.tdmClass](#) [tdmClassifyLoop](#) [tdmRegressLoop](#)

### Examples

```
### 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]
d_train=iris[idx_train,]
d_vali=iris[-idx_train,]
d_dis=iris[numeric(0),]
res <- tdmClassify(d_train,d_vali,d_dis,NULL,response.variables,input.variables,opts)

cat("\n")
print(res$allEVAL)
```

**Description**

tdmClassifyLoop contains a double loop (opts\$NRUN and CV-folds) and calls `tdmClassify`. It is called by all classification R-functions `main_*`.  
 It splits - if `tset` is `NULL` - the data in `dset` into training and validation data according to `opts$TST.kind`.  
 It returns an object of class `TDMclassifier`.

**Usage**

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

**Arguments**

<code>dset</code>	the data frame containing training and validation data.
<code>response.variables</code>	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)
<code>input.variables</code>	vector with names of input columns
<code>opts</code>	a list from which we need here the following entries <code>NRUN</code> number of runs (outer loop) <code>TST.SEED =NULL</code> : get a new random number seed with <code>tdmRandomSeed</code> . =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 <code>TST.kind=="cv"</code> or <code>"rand"</code> ) (see also <code>MOD.SEED</code> in <code>tdmClassify</code> ) <code>TST.kind</code> how to create <code>cvi</code> , handed over to <code>tdmModCreateCVindex</code> . If <code>TST.kind=="col"</code> , then <code>cvi</code> is taken from <code>dset[,opts\$TST.col]</code> . <code>GD.RESTART [TRUE] =TRUE/FALSE</code> : do/don't restart graphic devices <code>GD.DEVICE ["non" "win" "pdf" "png"]</code>
<code>tset</code>	[ <code>NULL</code> ] If not <code>NULL</code> , this is the test data set. If <code>NULL</code> , we are in tuning and the validation data set is build from <code>dset</code> according to the procedure prescribed in <code>opts\$TST.*</code> .

**Value**

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

<code>lastRes</code>	last run, last fold: result from <code>tdmClassify</code>
<code>C_train</code>	classification error on training set
<code>G_train</code>	gain on training set
<code>R_train</code>	relative gain on training set (percentage of max. gain on this set)
<code>*_vali</code>	— similar, with <code>vali</code> set instead of training set —
<code>*_vali2</code>	— similar, with <code>vali2</code> set instead of training set —
<code>Err</code>	a data frame with as many rows as <code>opts\$NRUN</code> 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.  
**predictTest** predictions on the test set tset (NULL if tset==NULL )  
**predProbList** a list, predProbList[[i]] has 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@th-koeln.de>), THK

### See Also

[print.TDMclassifier](#), [tdmClassify](#), [tdmRegress](#), [tdmRegressLoop](#)

### Examples

```

### ----- 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
opts$TST.SEED <- opts$MOD.SEED <- 5 # reproducible results
#opts$VERBOSE <- opts$SRF.verbose <- 0 # no printed output
gdObj <- 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)

```

---

tdmClassifySummary	<i>Print summary output for result from tdmClassifyLoop and add result\$y.</i>
--------------------	--

---

**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 <a href="#">tdmClassifyLoop</a> , 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	<i>Default values for list tdm.</i>
-----------------	-------------------------------------

---

**Description**

This list controls the tuning and unbiased evaluation phase. 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 <a href="#">tdmStartSpot2</a> and <a href="#">unbiasedRun</a>
unbiasedFunc	["unbiasedRun"] which function to call for unbiased evaluation
tuneMethod	["spot"] other choices: "cmaes", "bfgs", ..., see <a href="#">tdmDispatchTuner</a>
nExperim	[1]
umode	["RSUB"], one out of [ "RSUB"   "CV"   "TST"   "SP_T" ], see <a href="#">unbiasedRun</a>
timeMode	[1] 1: proc time, 2: system time, 3: elapsed time (columns Time.TST and Time.TRN in envT\$theFinals)
filenameEnvT	filename where <a href="#">tdmBigLoop</a> will save a small version of environment envT. If NULL, it is set to sub(".conf", ".RData", tdm\$runList[1]).
theSpotPath	[NA] use SPOT's package version
parallelCPUs	[1] 1: sequential, >1: parallel execution with this many CPUs (package <a href="#">parallel</a> )
parallelFuncs	[NULL] in case tdm\$parallelCPUs>1: a string vector with functions which are clusterExport'ed in addition to tdm\$mainFunc.
path	[NULL] from where to load and save envT resp. filenameEnvT. If it is NULL, tdm\$path is set to the actual working directory at the time when tdmEnvTMakeNew is executed.
runList	[NULL] a list of configuration names .conf
stratified	[NULL] see <a href="#">tdmReadAndSplit</a>
tdmPath	[NULL] from where to source the R sources. If NULL load library TDMR instead.
test2.string	["default cutoff"]
optsVerbosity	[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	[TRUE] if TRUE, save the last model, which is trained in unbiasedRun, onto filenameEnvT
tstCol	["TST.COL"] 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 <a href="#">cma_jTuner</a> .
CMA.populationSize	[NULL] (only for CMA-ES Java tuner) see <a href="#">cma_jTuner</a> .

**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 constructed. 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, THK, Patrick Koch

---

tdmEnvTAddBstRes	<i>Add BST and RES data frames to an existing envT environment.</i>
------------------	---

---

**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**

<code>envT</code>	the TDMR environment
<code>fileRData</code>	string with filename to load. This file is searched in <code>envT\$tdm\$path</code> .

**Value**

the augmented envT

---

tdmEnvTAddGetters	<i>Add getter functions getBst and getRes to environment envT</i>
-------------------	---

---

**Description**

Add getter functions getBst and getRes to environment envT

**Usage**

```
tdmEnvTAddGetters(envT)
```

**Arguments**

envT                    the TDMR environment

**Value**

the augmented envT

---

tdmEnvTGetOpts	<i>Return list opts from the k-th element of envT\$sCList</i>
----------------	---

---

**Description**

Return list opts from the k-th element of envT\$sCList

**Usage**

```
tdmEnvTGetOpts(envT, k = 1)
```

**Arguments**

envT                    environment TDMR  
k                        [1] index 1,...,length(envT\$runList)

**Value**

opts



---

tdmEnvTLoad	<i>Load an envT-type environment from file fileRData.</i>
-------------	---

---

### Description

The loaded envT is augmented with getter functions, see [tdmEnvTAddGetters](#).

### Usage

```
tdmEnvTLoad(fileRData, path = NULL)
```

### Arguments

fileRData	string with filename to load.
path	[NULL] dir where to search fileRData. If NULL, use current dir.

### Value

envT

---

tdmEnvTMakeNew	<i>Construct a new environment envT of class TDMenvir.</i>
----------------	--

---

### Description

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

### Usage

```
tdmEnvTMakeNew(tdm = NULL, sCList = defaultSCList())
```

### Arguments

tdm	a list with general settings for TDMR, see <a href="#">tdmDefaultsFill</a>
sCList	[defaultSC()] a list of list with controls for SPOT or other tuners (one list for each element in tdm\$runList)

**Value**

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

<code>runList</code>	<code>=tdm\$runList</code>
<code>tdm</code>	<code>=tdmDefaultsFill(tdm)</code>
<code>getBst</code>	accessor function( <code>confFile,nExp,theTuner</code> ) into <code>envT\$bstGrid</code>
<code>getRes</code>	accessor function( <code>confFile,nExp,theTuner</code> ) into <code>envT\$resGrid</code>
<code>sCList</code>	list of <code>spotConfig</code> -objects, as many as <code>envT\$runList</code> has elements. Each <code>spotConfig</code> object <code>sCList[[k]]</code> contains a list <code>opts</code> as element for the machine learning part.

**Author(s)**

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

**See Also**

[tdmBigLoop](#)

---

<code>tdmEnvTReport</code>	<i>Make a report plot based on envT</i>
----------------------------	---

---

**Description**

Given the results from a prior tuning run in `envT`, make a sensitivity plot for this run. If `envT$tdm$nrn > 0` then make additionally with the best-performing parameters from the tuning run a new unbiased run on the test data.

**Usage**

```
tdmEnvTReport(envT, ind)
```

**Arguments**

<code>envT</code>	results from a prior tuning run.
<code>ind</code>	an integer from <code>1:length(envT\$bstGrid)</code> : Take the tuning run with index <code>ind</code> .

**Value**

`envT`, with data frame `finals` added, if `envT$tdm$nrn > 0`.

**See Also**

[tdmEnvTReportSens](#)

## Examples

```
## The best results are read from demo02sonar/demoSonar.RData relative to the TDMR
## package directory.
path = paste(find.package("TDMR"), "demo02sonar", sep="/");
envT = tdmEnvTLoad("demoSonar.RData", path); # loads envT
source(paste(path, "main_sonar.r", sep="/"));
envT$tdm$nrnrun=0; # =0: don't, >0: do unbiasedRun with opts$NRUN=envT$tdm$nrnrun
envT$sCList[[1]]$opts$VERBOSE=1;
envT <- tdmEnvTReport(envT, 1);
if (!is.null(envT$theFinals)) print(envT$theFinals);
```

---

tdmEnvTReportSens      *Function to generate a report with sensitivity plot.*

---

## Description

The sensitivity curves are based on a metamodel which is a random forest with 100 trees fitted to the result points from RES-file. The plot contains: x-axis: ROI for each parameter normalized to [-1,1] y-axis:

## Usage

```
tdmEnvTReportSens(spotConfig)
```

## Arguments

spotConfig      the configuration list of all spot parameters

## See Also

[tdmEnvTReport](#)

---

tdmEnvTSetOpts      *Set list opts for the k-th element of envT\$sCList*

---

## Description

Set list opts for the k-th element of envT\$sCList

## Usage

```
tdmEnvTSetOpts(envT, opts, k = 1)
```

**Arguments**

envT	environment TDMR
opts	list of options
k	[1] index 1,...,length(envT\$runList)

**Value**

envT

---

tdmEnvTUpdate      *Update envT\$tdm*

---

**Description**

Update envT\$tdm with the non-NULL elements of tdm

**Usage**

```
tdmEnvTUpdate(envT, tdm)
```

**Arguments**

envT	environment TDMR
tdm	list for TDMR, see <a href="#">tdmDefaultsFill</a>

**Value**

envT

---

tdmGraAndLogFinalize      *Finalize graphics and log file*

---

**Description**

Finalize graphics and log file

**Usage**

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

**Arguments**

opts	with opts\$GD.DEVICE one out of [ "pdf"   "png"   "win"   "rstudio"   "non" ], see <a href="#">tdmGraphicInit</a>
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)

---



*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)

---



*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

opts	with opts\$GD.DEVICE one out of [ "pdf"   "png"   "win"   "rstudio"   "non" ], see <a href="#">tdmGraphicInit</a>
...	optional arguments (currently not used)

---

tdmGraphicInit     *Initialize graphic device.*

---

### Description

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

### Usage

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

### Arguments

opts	with opts\$GD.DEVICE one out of [ "pdf"   "png"   "win"   "rstudio"   "non" ] "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
...	optional arguments to hand over to <a href="#">pdf</a> (the other devices require no further arguments)

---

tdmGraphicNewWin	<i>Initialize a new window.</i>
------------------	---------------------------------

---

### Description

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

### Usage

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

### Arguments

opts	with opts\$GD.DEVICE one out of [ "pdf"   "png"   "win"   "rstudio"   "non" ], see <a href="#">tdmGraphicInit</a>
...	optional arguments to hand over to <a href="#">png</a> or windows or X11 in package grDevices (the other devices require no further arguments)

---

tdmGraphicToTop	<i>Bring the active window to the top</i>
-----------------	---

---

### Description

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

### Usage

```
tdmGraphicToTop(opts)
```

### Arguments

opts	with opts\$GD.DEVICE one out of [ "pdf"   "png"   "win"   "rstudio"   "non" ], see <a href="#">tdmGraphicInit</a>
------	---

---

tdmMapDesApply	<i>Apply the mapping from des to opts.</i>
----------------	--

---

### Description

For each variable which appears in .roi (and thus in 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 [defaultOpts](#).

### Usage

```
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 envT\$mapUser, see <a href="#">tdmMapDesLoad</a> , 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	<i>Load the mapping files.</i>
---------------	--------------------------------

---

### 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

```
tdmMapDesLoad(tdm = list())
```



**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](#)

---

tdmModConfmat	<i>Calculate confusion matrix, gain and RGain measure.</i>
---------------	--

---

**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: <ul style="list-style-type: none"> <li>• 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)</li> <li>• rgain.type: one out of {"rgain"   "meanCA"   "minCA"   "bYouden"   "arROC"   "arLIFT"   "arPRE" }, affects output cm\$mat and cm\$rgain, see below.</li> </ul>
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=="ar*".

**Value**

cm, a list containing:

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

**Note**

For all measures rgain holds: The higher, the better.

The last four elements of opts\$rgain.type= "bYouden", "arROC", "arLIFT", "arPre" are only available for binary classification.

For case "bYouden":

sensitivity = TP/(TP+FN)

specificity = TN/(TN+FP)

**Author(s)**

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

**See Also**

[tdmClassify](#) [tdmROCRbase](#)

---

tdmModCreateCVindex     *Create and return a training-validation-set index vector.*

---

## 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.

In summary, TST.kind="cv" means cross validation (TST.NFOLD models are built with TST.NFOLD different train-validation data sets), while TST.kind="rand" or "col" means one model build with a random ("rand") or user-defined ("col") training-validation split.

## Usage

```
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 <ul style="list-style-type: none"> <li>• TST.kind: ["cv" "rand" "col"]</li> <li>• TST.NFOLD: number of CV folds (only relevant in case TST.kind=="cv")</li> <li>• TST.COL: column of dset containing the (0/1/&lt;0) index (only relevant in case TST.kind=="col") or NULL if no such column exists</li> <li>• TST.valiFrac: fraction of records to set aside for validation (only relevant in case TST.kind=="rand")</li> <li>• TST.trnFrac: [1-opts\$TST.valiFrac] fraction of records to use for training (only relevant in case TST.kind=="rand")</li> </ul>
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.)

---

tdmModSortedRFimport *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
response.variable	the target column from d_train to use for the RF-model
input.variables	the input columns from d_train to use for the RF-model
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  $\geq 0$ , 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.minlsi: [1] a lower bound for the length of SRF\$input.variables
- RF.sampsize: sampsize for RF, set prior to calling this func via tdmModAdjustSampsize(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@th-koeln.de>

---

tdmModVote2Target	<i>Analyze how the vote fraction corresponds to reliability of prediction.</i>
-------------------	--

---

## 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(vote0, pred, target)
```

## Arguments

vote0	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]]
pred0	fraction of 0-predictions
pCorr	fraction of correct predictions
pR	fraction of true 1-cases

**Author(s)**

Wolfgang Konen <wolfgang.konen@th-koeln.de>

---

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:

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

**Usage**

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

**Arguments**

<code>opts</code>	(optional) the options already set
<code>path</code>	["."] where to find everything for the DM task.

## Details

The path-related settings are relative to `opts$path`, 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

<code>path</code>	<code>["."] where to find everything for the DM task</code>
<code>dir.txt</code>	<code>[data] where to find .txt/.csv files</code>
<code>dir.data</code>	<code>[data] where to find other data files, including .Rdata</code>
<code>dir.output</code>	<code>[Output] where to put output files</code>
<code>filename</code>	<code>["default.txt"] the task data</code>
<code>filetest</code>	<code>[NULL] the test data, only relevant for READ.TstFn!=NULL</code>
<code>data.title</code>	<code>["Default Data"] title for plots</code>
<code>READ.TXT</code>	<code>[T] =T: read data from .csv and save as .Rdata, =F: read from .Rdata</code>
<code>READ.NROW</code>	<code>[-1] read this amount of rows or -1 for 'read all rows'</code>
<code>READ.TrnFn</code>	<code>function to be passed into <a href="#">tdmReadDataset</a>. Signature: function(opts) returning a data frame. It reads the train-validation data.</code>
<code>READ.TstFn</code>	<code>[NULL] function to be passed into <a href="#">tdmReadDataset</a>. Signature: function(opts) returning a data frame. It reads a separate test data file. If NULL, this reading step is skipped.</code>
<code>READ.INI</code>	<code>[TRUE] read the task data initially, i.e. prior to tuning, using <a href="#">tdmReadDataset</a>. If =FALSE, the data are read anew in each pass through main_TASK, i.e. in each tuning step (deprecated).</code>
<code>TST.kind</code>	<code>["rand"] one of the choices from {"cv","rand","col"}, see <a href="#">tdmModCreateCVindex</a> for details</code>
<code>TST.COL</code>	<code>["TST.COL"] name of column with train/test/disregard-flag</code>
<code>TST.NFOLD</code>	<code>[3] number of CV-folds (only for TST.kind=="cv")</code>
<code>TST.valiFrac</code>	<code>[0.1] set this fraction of the train-validation data aside for validation (only for TST.kind=="rand")</code>
<code>TST.testFrac</code>	<code>[0.1] set prior to tuning this fraction of data aside for testing (if <code>tdm\$umode=="SP_T"</code> and <code>opts\$READ.INI==TRUE</code>) or set this fraction of data aside for testing after tuning (if <code>tdm\$umode=="RSUB"</code> or <code>=="CV"</code>)</code>
<code>TST.trnFrac</code>	<code>[NULL] train set fraction, if NULL then <a href="#">tdmModCreateCVindex</a> will set it to 1 - <code>opts\$TST.valiFrac</code>.</code>
<code>TST.SEED</code>	<code>[NULL] a seed for the random test set selection (<a href="#">tdmRandomSeed</a>) and random validation set selection. (<a href="#">tdmClassifyLoop</a>). If NULL, use <a href="#">tdmRandomSeed</a>.</code>
<code>PRE.PCA</code>	<code>["none" (default)"linear"] PCA preprocessing: [don't   do normal PCA (prcomp)]</code>

PRE.PCA.REPLACE	[T] =T: replace with the PCA columns the original numerical columns, =F: add the PCA columns
PRE.PCA.npc	[0] if >0: add monomials of degree 2 from the first PRE.PCA.npc columns (PCs) (only active, if opts\$PRE.PCA!="none")
PRE.SFA	["none" (default)   "2nd"] SFA preprocessing (see package <a href="#">rSFA</a> : [don't   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 T: don't   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 <a href="#">sfaPBootstrap</a> in package <a href="#">rSFA</a> for its interface description
PRE.allNonValid	[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 <a href="#">tdmPreGroupLevels</a> )
PRE.MaxLevel	[32] bind the N-32+1 least frequent cases in column OTHER (see <a href="#">tdmPreGroupLevels</a> )
SRF.kind	["xperc" (default)   "ndrop"   "nkeep"   "none" ] the method used for feature selection, see <a href="#">tdmModSortedRFimport</a>
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	[T] =T: calculate importance & save on SRF.file, =F: load from srfFile (srfFile = Output/<confFile>.SRF.Rdata)
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 SRF\$input.variables
SRF.method	["RFimp"]
SRF.scale	[TRUE] option 'scale' for call importance() in <a href="#">tdmModSortedRFimport</a>



MOD.SEED	[NULL] a seed for the random model initialization (if model is non-deterministic). If NULL, use <a href="#">tdmRandomSeed</a> .
MOD.method	["RF" (default)   "MC.RF"   "SVM"   "NB" ]: use [RF   MetaCost-RF   SVM   Naive Bayes ] in <a href="#">tdmClassify</a> ["RF" (default)   "SVM"   "LM" ]: use [RF   SVM   linear model ] in <a href="#">tdmRegress</a>
RF.ntree	[500]
RF.samp	[1000] 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 <a href="#">tdmModAdjustSampsize*</a> .
RF.mtry	[NULL]
RF.nodesize	[1]
RF.OOB	[TRUE] if =T, return OOB-training set error as tuning measure; if =F, return validation set error
RF.p.all	[FALSE]
SVM.kernel	[3] =1: linear, =2: polynomial, =3: RBF, =4: sigmoid
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.coeflearn	[1]=1: "Breiman", =2: "Freund", =3: "Zhu" as value for boosting(...,coeflearn,...) (AdaBoost)
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 <a href="#">tdmClassify</a> . 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 <a href="#">tdmClassify</a>

rgain.type	["rgain" (default)   "meanCA"   "minCA" ] in case of <code>tdmClassify</code> : For classification, the measure Rgain returned from <code>tdmClassifyLoop</code> in <code>result\$R_*</code> is [relative gain (i.e. gain/gainmax)   mean class accuracy   minimum class accuracy   minus Y ]. The goal is to maximize Rgain. For binary classification there are the additional measures [ "arROC"   "arLIFT"   "arPRE"   "bYouden" ], see 'Value' in <code>tdmModConfmat</code> . For regression, the goal is to minimize <code>result\$R_*</code> returned from <code>tdmRegress</code> . In this case, possible values are <code>rgain.type = ["rmae" (default)   "rmse"   "made" ]</code> which stands for [ relative mean absolute error   root mean squared error   mean absolute deviation ].
ncopies	[0] if >0, activate <code>tdmParaBootstrap</code> in <code>tdmClassify</code>
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 <code>tdmClassify</code> if it is not NULL.
GD.DEVICE	["win"] ="win": all graphics to (several) windows (windows or X11 in package <code>grDevices</code> ) ="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 <code>opts\$GD.PNGDIR</code> ="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 <code>tdmClassify</code> or <code>tdmRegress</code> , 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 <code>main_*.r</code> (suitable for <code>main_*.r</code> solo) or =F: do not close (suitable for call from <code>tdmStartSpot2</code> , where all windows should remain open)
NRUN	[2] how many runs with different train & test samples - or - how many CV-runs, if <code>opts\$TST.kind="cv"</code>
APPLY_TIME	[FALSE]
test2.show	[FALSE]
test2.string	["default cutoff"]
VERBOSE	[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, THK, 2013 - 2018

**See Also**

[tdmOptsDefaultsFill](#) [tdmDefaultsFill](#)

---

tdmParaBootstrap	<i>Parametric bootstrap: add 'noisy copies' to a data frame (training data).</i>
------------------	--

---

**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**

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

---

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**

<code>dset</code>	the target data frame
<code>rx</code>	a data frame where to draw the monomials from
<code>PRE.npc</code>	the number of columns from <code>rx</code> to use (clipped to <code>ncol(rx)</code> if necessary)
<code>opts</code>	a list from which we need here the following entries: <ul style="list-style-type: none"> <li>• <code>filename</code></li> <li>• <code>VERBOSE</code></li> </ul>
<code>degree</code>	[2] (currently only 2 is supported)
<code>prefix</code>	["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	<i>Find constant columns.</i>
--------------------	-------------------------------

---

**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**

<code>dset</code>	data frame
-------------------	------------

**Value**

name vector of constant columns

---

tdmPreGroupLevels	<i>Group the levels of factor variable in <code>dset[, colname]</code>.</i>
-------------------	---

---

**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**

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

**Arguments**

<code>dset</code>	data frame
<code>colname</code>	name of column to be re-grouped
<code>opts</code>	list, here we need <ul style="list-style-type: none"> <li>• <code>PRE.Xpgroup</code> [0.99]</li> <li>• <code>PRE.MaxLevel</code> [32] (32 is the maximum number of levels allowed for <a href="#">randomForest</a>)</li> </ul>

**Value**

dset, 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**

```
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 <ul style="list-style-type: none"> <li>• opts\$thresh_pR</li> <li>• opts\$verbose</li> </ul>

**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	<i>Apply PCA (Principal Component Analysis) to new data.</i>
-----------------	--

---

**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**

<code>dset</code>	the data frame with the new data
<code>pcaList</code>	a value returned from a prior call to <code>tdmPrePCA.train</code>
<code>opts</code>	a list from which we need here the following entries: <ul style="list-style-type: none"> <li>• <code>PRE.knum</code>: if &gt;0 and if <code>PRE.PCA="kernel"</code>, take only a subset of <code>PRE.knum</code> records from <code>dset</code></li> <li>• <code>PRE.PCA.npc</code>: if &gt;0, then add for the first <code>PRE.PCA.npc</code> PCs the monomials of degree 2 (see <code>tdmPreAddMonomials</code>)</li> <li>• <code>PRE.PCA.numericV</code> vector with all column names in <code>dset</code> for which PCA is performed. These columns may contain <i>*numeric*</i> values only.</li> </ul>
<code>dtrain</code>	[NULL] optional, only needed in case that <code>dset</code> is a 0-row-data frame: then we 'borrow' the columns from <code>dtrain</code> , the data set returned from <code>tdmPrePCA.train</code> in <code>pca\$dset</code> .

**Value**

`pca`, a list with entries:

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

**Author(s)**

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

**See Also**

[tdmPrePCA.train](#)

---

tdmPrePCA.train	<i>PCA (Principal Component Analysis) for numeric columns in a data frame.</i>
-----------------	--

---

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

### 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 <a href="#">prcomp</a> plus eival, 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.



**Author(s)**

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

**See Also**

[tdmPrePCA.apply](#)

---

tdmPreSFA.apply	<i>Apply SFA (Slow Feature Analysis) to new data.</i>
-----------------	---

---

**Description**

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

**Usage**

```
tdmPreSFA.apply(dset, sfaList, opts, dtrain = NULL)
```

**Arguments**

<code>dset</code>	the data frame with the new data
<code>sfaList</code>	a value returned from a prior call to <code>tdmPreSFA.train</code>
<code>opts</code>	a list from which we need here the following entries: <ul style="list-style-type: none"> <li>• <code>PRE.SFA.REPLACE</code>: [T]=T: replace the original numerical columns with the SFA columns; =F: add the SFA columns</li> <li>• <code>PRE.SFA.npc</code>: if &gt;0, then add for the first <code>PRE.SFA.npc</code> PCs the monomials of degree 2 (see <code>tdmPreAddMonomials</code>)</li> <li>• <code>PRE.SFA.ODIM</code>: [5] number of SFA output dimensions (slowest signals) to return</li> <li>• <code>PRE.SFA.numericV</code> vector with all column names in <code>dset</code> for which SFA is performed. These columns may contain <code>*numeric*</code> values only.</li> </ul>
<code>dtrain</code>	[NULL] optional, only needed in case that <code>dset</code> is a 0-row-data frame: then we 'borrow' the columns from <code>dtrain</code> , the data set returned from <code>tdmPreSFA.train</code> in <code>sfa\$dset</code> .

**Value**

`sfa`, a list with entries:

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

**Author(s)**

Wolfgang Konen, Martin Zaeferrer, FHK, Jan'2012 - Feb'2012

**See Also**

[tdmPreSFA.train](#)

---

tdmPreSFA.train	<i>SFA (Slow Feature Analysis) for numeric columns in a data frame.</i>
-----------------	---

---

**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**

dset	the data frame with training (and test) data.
response.var	the response variable for classification.
opts	a list from which we need here the following entries: <ul style="list-style-type: none"> <li>• PRE.SFA: [ "linear"   "2nd"   "none" ] which stands for [ 1st   2nd degree monomial SFA   no SFA ]</li> <li>• PRE.SFA.REPLACE: [T] =T: replace the original numerical columns with the SFA columns; =F: add the SFA columns</li> <li>• PRE.SFA.npc: if &gt;0, then add for the first PRE.SFA.npc PCs the monomials of degree 2 (see <code>tdmPreAddMonomials</code>)</li> <li>• PRE.SFA.PPRANGE: [11] number of inputs after preprocessing, they enter into expansion</li> <li>• PRE.SFA.ODIM: [5] number of SFA output dimensions (slowest signals) to return</li> <li>• PRE.SFA.numericV vector with all column names in dset which are input for SFA. These columns may contain <i>*numeric*</i> values only.</li> </ul>

**Value**

sfa, a list with entries:

dset	the input data frame dset with columns <code>numeric.variables</code> replaced or extended (depending on <code>opts\$PRE.SFA.REPLACE</code> ) by the SFA components with names SF1, SF2, ... and with optional monomial columns added, if <code>PRE.SFA.npc&gt;0</code>
------	---

`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` (`sfaOpts`), matrices `DSF` and `SF` and many others, as returned from [sfaStep](#)

**Author(s)**

Wolfgang Konen, Martin Zaefferer, FHK, Jan'2012 - Feb'2012

**See Also**

[tdmPreSFA.apply](#)

---

tdmRandomSeed	<i>Generates pseudo-random random number seeds.</i>
---------------	---

---

**Description**

To use this mechanism, create first an object `tdmRandomSeed` with a call to [makeTdmRandomSeed](#).

**Usage**

```
tdmRandomSeed()
```

**Value**

In each call to this function a different integer in  $0 \dots 100001 + nCall$  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@th-koeln.de>

**See Also**

[makeTdmRandomSeed](#)

**Examples**

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

---

tdmReadAndSplit	<i>Read and split the task data.</i>
-----------------	--------------------------------------

---

### Description

Read the task data using [tdmReadDataset](#) and split them into a test part and a training/validation-part and return a [TDMdata](#) object.

### Usage

```
tdmReadAndSplit(opts, tdm, nExp = 0, dset = NULL)
```

### Arguments

opts	<p>a list from which we need here the elements</p> <ul style="list-style-type: none"> <li>• READ.INI: [T] =T: do read and split, =F: return NULL</li> <li>• READ.*: other settings for <a href="#">tdmReadDataset</a></li> <li>• filename: needed for <a href="#">tdmReadDataset</a></li> <li>• filetest: needed for <a href="#">tdmReadDataset</a></li> <li>• TST.testFrac: [0.1] set this fraction of the daa aside for testing</li> <li>• 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.)</li> </ul>
tdm	<p>a list from which we need here the elements</p> <ul style="list-style-type: none"> <li>• mainFile: if not NULL, set working dir to dir(mainFile) before executing <a href="#">tdmReadDataset</a></li> <li>• umode: [ "RSUB"   "CV"   "TST"   "SP_T" ], how to divide in training/validation data for tuning and test data for the unbiased runs</li> <li>• SPLIT.SEED: if NULL, set random number generator (RNG) to <a href="#">tdmRandomSeed</a> when constructing. dataObj. If not NULL, set RNG to SPLIT.SEED + nExp -&gt; deterministic test set split</li> <li>• stratified: [NULL] string specifying the column with the response variable for classification. 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</li> </ul>
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.

### Details

If dset is NULL, the files specified in opts are read into dset, see [tdmReadDataset](#) for details. Then, depending on the value of tdm\$umode

- "SP\_T": split the data randomly into training and test data with test set fraction according to opts\$TST.testFrac. Make use of tdm\$SPLIT.SEED and tdm\$stratified, if given. Set TST.COL to "tdmSplit".

- "RSUB", "CV": use all data for training/validation. That is, the training-validation split is done later in [tdmClassifyLoop](#) or [tdmRegressLoop](#).
- "TST": split the data into training and test data according to column. `opts$TST.COL` (usually "TST.COL"), which carries a 1 for each test record and a 0 else. If `opts$filetest` is specified, then all records from this file will carry a 1 in `opts$TST.COL`. All records from `opts$filename` carry a 0.

### Value

`dataObj`, either NULL (if `opts$READ.INI==FALSE`) or an object of class [TDMdata](#) containing

<code>dset</code>	a data frame with the complete data set
<code>TST.COL</code>	string, the name of the column in <code>dset</code> which has a 1 for records belonging to the test set and a 0 for train/vali records. If <code>tdm\$umode=="SP_T"</code> , then <code>TST.COL="tdmSplit"</code> , else <code>TST.COL=opts\$TST.COL</code> .
<code>filename</code>	<code>opts\$filename</code> , 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@th-koeln.de](mailto:wolfgang.konen@th-koeln.de)>), THK

### See Also

[dsetTrnVa.TDMdata](#), [dsetTest.TDMdata](#), [tdmReadDataset](#), [tdmBigLoop](#)

---

<code>tdmReadDataset</code>	<i>Read data according to opts settings.</i>
-----------------------------	--

---

### Description

Read the data according to the settings `opts$READ.*` and `opts$TST.COL` (see Details).

### Usage

```
tdmReadDataset(opts)
```

**Arguments**

- `opts` list of options, we need here
- `READ.TrnFn`: [`tdmReadTrain`] function with argument `opts` for reading the training data and returning them in a data frame
  - `READ.NROW`: [`-1`] read only that many rows from `opts$filename`. `-1` for 'read all rows'.
  - `READ.TstFn`: [`NULL`] function with argument `opts` for reading the test data and returning them in a data frame. If `NULL` then skip test file reading.
  - `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
  - `path`: used by `READ.TrnFn` to locate file
  - `dir.data`: used by `READ.TrnFn` to locate file

**Details**

When `opts$READ.TstFn==NULL`, then only `opts$READ.TrnFn` is used.

When `opts$READ.TstFn!=NULL`, the following things happen in `tdmReadDataset`: Data are read from `opts$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

**See Also**

[tdmReadAndSplit](#)

---

<code>tdmReadTaskData</code>	<i>Read task data.</i>
------------------------------	------------------------

---

**Description**

Read and split task data (wrapper for [tdmReadAndSplit](#)).

**Usage**

```
tdmReadTaskData(envT, tdm)
```

**Arguments**

- `envT` environment TDMR
- `tdm` list with TDMR controls

**Value**

dataObj, see [tdmReadAndSplit](#)

**See Also**

[dsetTrnVa.TDMdata](#), [dsetTest.TDMdata](#), [tdmReadAndSplit](#)

---



*Core regression function of TDMR.*


---

**Description**

tdmRegress is called by [tdmRegressLoop](#) and returns an object of class `tdmRegre`. 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**

```
tdmRegress(d_train, d_test, d_preproc, response.variables, input.variables,
           opts, tsetStr = c("Validation", "validation", ".vali"))
```

**Arguments**

<code>d_train</code>	training set
<code>d_test</code>	test set, same columns as training set
<code>d_preproc</code>	data used for preprocessing. May be NULL, if no preprocessing is done ( <code>opts\$PRE.SFA=="none"</code> and <code>opts\$PRE.PCA=="none"</code> ). If preprocessing is done, then <code>d_preproc</code> is usually all non-validation data.
<code>response.variables</code>	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)
<code>input.variables</code>	vector with names of input columns
<code>opts</code>	additional parameters [defaults in brackets] SRF.* several parameters for sorted_rf_importance (see <code>tdmModelingUtils.r</code> ) 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 =NULL: 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  
 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  
 VERBOSE [2] =2: most printed output, =1: less, =0: no output  
 tsetStr [c("Validation", "validation", ".vali")]

### 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

```

### 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 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, tset = NULL)
```

## 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 <ul style="list-style-type: none"> <li>NRUN number of runs (outer loop)</li> <li>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 <a href="#">tdmClassify</a>)</li> <li>TST.kind how to create cvi, handed over to <a href="#">tdmModCreateCVindex</a>. If TST.kind="col", then cvi is taken from dset[,opts\$TST.col].</li> </ul>

```

GD.RESTART [TRUE] =TRUE/FALSE: do/don't restart graphic devices
GRAPHDEV ["non"| other ]
tset      [NULL] If not NULL, this is the test data set. If NULL, we are in tuning and
          the validation data set is build from dset according to the procedure prescribed
          in opts$TST.*.

```

**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.
predictTest predictions on the test set tset (NULL if tset==NULL )

```

**Author(s)**

Wolfgang Konen (<wolfgang.konen@th-koeln.de>), THK

**See Also**

[tdmRegress](#), [tdmClassifyLoop](#), [tdmClassify](#)

**Examples**

```

### ----- 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$NRUN=2 times with different random train-validation set splits.
### Therefore data frame result$Err has 2 rows.
###
opts=tdmOptsDefaultsSet()           # set all defaults for data mining process
gdObj <- 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)

```

---

tdmRegressSummary	<i>Print summary output for result from tdmRegressLoop and add result\$y.</i>
-------------------	---

---

### 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

```
tdmRegressSummary(result, opts, dset = NULL)
```

### Arguments

result	return value from a prior call to <a href="#">tdmRegressLoop</a> , 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
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 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

```
## 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: For 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

```
## Not run:
path <- paste(find.package("TDMR"), "demo02sonar", sep="/");
source(paste(path,"main_sonar.r", sep="/"));
result = main_sonar();
tdmROCR(result);

## End(Not run)
```

---

tdmROCRbase	<i>Single plot of ROC, lift or other chart for a <code>TDMclassifier</code> object.</i>
-------------	---

---

## Description

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

## Usage

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

## Arguments

<code>x</code>	return value from a prior call to <code>tdmClassifyLoop</code> , an object of class <code>TDMclassifier</code> .
<code>dataset</code>	["validation"] which part of the data to use, either "training" or "validation"
<code>nRun</code>	[1] if <code>x</code> contains multiple runs, which run to show (1,..., <code>Opts(x)\$NRUN</code> )
<code>typ</code>	["ROC"] which chart type, one out of ("ROC", "lift", "precRec") for (ROC, lift, precision-recall)-chart (see <a href="#">performance</a> in package <code>ROCR</code> for more details): <ul style="list-style-type: none"> <li>• "ROC": receiver operating curve, TPR vs. FPR, with <math>TPR = TP / (TP + FN) = TP / P</math> and <math>FPR = FP / (FP + TN) = FP / N</math> (true and false positive rate).</li> <li>• "lift": lift chart, LIFT vs. RPP, with <math>LIFT = TPR / RPR</math> with random positive rate <math>RPR = P / (P + N)</math> and <math>RPP = (TP + FP) / (P + N)</math> (rate of pos. predictions).</li> <li>• "precRec": precision-recall-chart, PREC vs. RECALL, with <math>PREC = TP / (TP + FP)</math> and <math>RECALL = TP / P</math> (same as TPR).</li> </ul>
<code>noPlot</code>	[FALSE] if TRUE, suppress the plot, return only the area under curve
<code>...</code>	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

```
##*# ----- demo/demo05ROCR.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 validataion data set and
##*#           plot lift chart for training data set
##*#
```

```

path <- paste(find.package("TDMR"), "demo02sonar", sep="/");
#path <- paste("../inst", "demo02sonar", sep="/");

source(paste(path,"main_sonar.r", sep="/")); # defines readTrnSonar

controlDM <- function() {
  #
  # settings for the DM process (former sonar_00.apd file):
  # (see ?tdmOptsDefaultsSet for a complete list of all default settings
  # and many explanatory comments)
  #
  opts = list(path = path,
              dir.data = "data", # relative to path
              filename = "sonar.txt",
              READ.TrnFn = readTrnSonar, # defined in main_sonar.r
              data.title = "Sonar Data",
              NRUN = 1,
              rgain.type = "arROC",
              VERBOSE = 2
            );
  opts <- setParams(opts, defaultOpts(), keepNotMatching = TRUE);
}

opts <- controlDM();
result <- main_sonar(opts);

tdmGraphicNewWin(opts);
cat("Area under ROC-curve for validation data set: ",
    tdmROCRbase(result), "\n"); # side effect: plot ROC-curve
tdmGraphicNewWin(opts);
cat("Area under lift curve for training data set: ",
    tdmROCRbase(result, dataset="training", typ="lift"), "\n"); # side effect: plot lift chart

```

---

tdmTuneIt

*Tuning and unbiased evaluation (single tuning).*


---

## Description

For the first configuration name `.conf` in `tdm$runList` call the first tuning algorithm in `tdm$tuneMethod` (via function `tdmDispatchTuner`). After tuning perform with the best parameters a run of `tdm$unbiasedFunc` (usually `unbiasedRun`).

This experiment is repeated `tdm$nExperiment` times.

## Usage

```
tdmTuneIt(envT, dataObj)
```

**Arguments**

envT	an environment containing on input at least the element <code>tdm</code> (a list with general settings for TDMR, see <a href="#">tdmDefaultsFill</a> ), which has at least the elements <code>tdm\$runList</code> list of configuration names <code>.conf</code> <code>tdm\$tuneMethod</code> the tuner
dataObj	object of class <a href="#">TDMdata</a> (constructed here with the help of <a href="#">tdmReadAndSplit</a> ).

**Details**

`tdmTuneIt` differs from [tdmBigLoop](#) in that it processes only one configuration `.conf` and that it has `dataObj` as a mandatory calling parameter. This simplifies the data flow and is thus less error-prone.

`tdm` refers to `envT$tdm`.

See Details in [tdmBigLoop](#) for the list of available tuners.

**Value**

environment <code>envT</code> ,	containing the results
<code>res</code>	data frame with results from last tuning (one line for each call of <code>tdmStart*</code> )
<code>bst</code>	data frame with the best-so-far results from last tuning (one line collected after each (SPO) step)
<code>resGrid</code>	list with data frames <code>res</code> from all tuning runs. Use <code>envT\$getRes(envT, confFile, nExp, theTuner)</code> to retrieve a specific <code>res</code> .
<code>bstGrid</code>	list with data frames <code>bst</code> from all tuning runs. Use <code>envT\$getBst(envT, confFile, nExp, theTuner)</code> to retrieve a specific <code>bst</code> .
<code>theFinals</code>	data frame with one line for each triple ( <code>confFile, nExp, tuner</code> ), each line contains summary information about the tuning run in the form: <code>confFile tuner nExp [params] NRUN NEVAL RGain.bst RGain.* sdR.*</code> where <code>[params]</code> is written depending on <code>tdm\$withParams</code> . <code>NRUN</code> is the number of unbiased evaluation runs. <code>NEVAL</code> is the number of function evaluations (model builds) during tuning. <code>RGain</code> 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 <code>RGain.*</code> is replaced by <code>RMAE.*</code> , the relative mean absolute error. Each <code>'sdR.'</code> denotes the standard deviation of the preceding <code>RGain</code> or <code>RMAE</code> . <code>RGain.bst</code> is the best result during tuning obtained on the training-validation data. <code>RGain.avg</code> is the average result during tuning. The following pairs <code>RGain.* sdR.*</code> are the results of one or several unbiased evaluations on the test data where <code>'*'</code> takes as many values as there are elements in <code>tdm\$umode</code> (the possible values are explained in <a href="#">unbiasedRun</a> ).
<code>result</code>	object of class <a href="#">TDMclassifier</a> or <a href="#">TDMregressor</a> . This is a list with results from <code>tdm\$mainFunc</code> 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 `tdmDispatchTuner`. 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:

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

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

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@th-koeln.de>), THK

### See Also

[tdmBigLoop](#), [tdmDispatchTuner](#), [unbiasedRun](#)

### Examples

```
### 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,
### http://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+%28Sonar,+Mines+vs.+Rocks%29).
### 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.

## path is the dir with data and main_*.r file:
path <- paste(find.package("TDMR"), "demo02sonar", sep="/");
#path <- paste("../..../inst", "demo02sonar", sep="/");

## control settings for TDMR
tdm <- list( mainFunc="main_sonar"
            , umode="CV" # { "CV" | "RSUB" | "TST" | "SP_T" }
            , tuneMethod = c("lhd")
            , filenameEnvT="exBigLoop.RData" # file to save environment envT
            , nrun=1, nfold=2 # repeats and CV-folds for the unbiased runs
            , nExperim=1
            , optsVerbosity = 0 # the verbosity for the unbiased runs
            );
source(paste(path,"main_sonar.r",sep="/")); # main_sonar, readTrnSonar

### This demo is for example and help (more meaningful, a bit higher budget)
source(paste(path,"control_sonar.r",sep="/")); # controlDM, controlSC

ctrlSC <- controlSC();
ctrlSC$opts <- controlDM();

#
# perform a complete tuning + unbiased eval
#
envT <- tdmEnvTMakeNew(tdm,sCList=list(ctrlSC)); # construct envT from settings in tdm and ctrlSC
dataObj <- tdmReadTaskData(envT,envT$tdm);
envT <- tdmTuneIt(envT,dataObj=dataObj); # start the tuning loop

```

---

unbiasedRun

*Perform unbiased runs with best-solution parameters.*


---

## 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, umode = "RSUB",
            withParams = FALSE, tdm = NULL)

```

**Arguments**

confFile	the configuration name, 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.
finals	[NULL] a one-row data frame to which new columns with final results are added
dataObj	[NULL] contains the pre-fetched data with training-set and test-set part. If NULL, set it to <code>tdmReadAndSplit(opts, tdm)</code> . It is now <b>deprecated</b> to have <code>dataObj==NULL</code> .
umode	— deprecated as argument to <code>unbiasedRun</code> —, use the division provided in <code>dataObj = tdmReadAndSplit(opts, tdm)</code> which makes use of <code>tdm\$umode</code> . For downward compatibility only (if <code>dataObj==NULL</code> ): ["RSUB" (default)   "CV"   "TST"   "SP_T"], how to divide in training and test data for the unbiased runs: "RSUB" random subsampling into $(1-\text{tdm}\$TST.\text{testFrac})\%$ training and $\text{tdm}\$TST.\text{testFrac}\%$ test data "CV" cross validation (CV) with <code>tdm\$nrun</code> folds "TST" all data in <code>opts\$filename</code> (or <code>dsetTrnVa(dataObj)</code> ) are used for training, all data in <code>opts\$filetest</code> (or <code>dsetTest(dataObj)</code> ) are used for testing "SP_T" 'split_test': prior to tuning, the data set was split by random subsampling into $\text{tdm}\$TST.\text{testFrac}\%$ test and $(1-\text{tdm}\$TST.\text{testFrac})\%$ training- <code>vali</code> data, tagged via column "tdmSplit". Tuning was done on training- <code>vali</code> data. Now we use column "tdmSplit" to select the test data for unbiased evaluation. Training during unbiased evaluation is done on a fraction <code>tdm\$\\$TST.trnFrac</code> of the training- <code>vali</code> data
withParams	[FALSE] if =TRUE, add columns with best parameters to data frame <code>finals</code> (should be FALSE, if different runs have different parameters)
tdm	a list with TDM settings from which we use here the elements <b>mainFunc</b> the function to be called for unbiased evaluations <b>mainFile</b> change to the directory of <code>mainFile</code> before starting <code>mainFunc</code> <b>nrun</b> [5] how often to call the unbiased evaluation <b>ifold</b> [10] how many folds in CV (only relevant for <code>umode="CV"</code> ) <b>TST.testFrac</b> [0.2] test set fraction (only relevant for <code>umode="RSUB"</code> or <code>"SP_T"</code> ) The defaults in '[...]' are set by <code>tdmDefaultsFill</code> , if they are not defined on input.

**Value**

envT the augmented environment envT, with the following items updated

finals	the final results
tdm	the updated list with TDM settings
results	last results (from last unbiased training)

**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`.

`spotConfig$opts` (list with all parameter settings for the DM task) has to be non-NULL.

**Author(s)**

Wolfgang Konen, THK, 2013 - 2018

If `envT$bst` or `envT$res` is NULL, try to read it from the file (the filename is inferred `envT$spotConfig`. If this is NULL, it is constructed from `confFile`). We try to find the files for `envT$bst` or `envT$res` in `dir envT$theTuner`).

**See Also**

[tdmBigLoop](#), [TDMclassifier](#), [TDMregressor](#)

**Examples**

```
## 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.
path = paste(find.package("TDMR"), "demo02sonar", sep="/")
envT = tdmEnvTLoad("demoSonar.RData",path); # loads envT
source(paste(path,"main_sonar.r",sep="/"));
envT$tdm$opts$verbosity=1;
envT$sCList[[1]]$opts$path=path; # overwrite a possibly older stored path
envT$spotConfig <- envT$sCList[[1]];
dataObj <- tdmReadTaskData(envT,envT$tdm);
envT <- unbiasedRun("sonar_04.conf",envT,dataObj,tdm=envT$tdm);
print(envT$finals);
```

# 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, 13
  
- cma\_es, 13
- cma\_jTuner, 13, 23
- cmaesTuner, 13
  
- defaultOpts, 4, 6, 12, 32
- defaultSC, 4, 5, 12
- dsetTest.TDMdata, 6, 7, 53, 55
- dsetTrnVa.TDMdata, 6, 7, 53, 55
  
- lhdTuner, 13
  
- makeTdmRandomSeed, 51
  
- Opts, 7, 60, 61
  
- parallel, 3, 22
- pdf, 30
- performance, 61
- png, 31
- prcomp, 48
- predict.TDMclassifier
  - (predict.TDMenvir), 8
- predict.TDMenvir, 8
- predict.TDMregressor
  - (predict.TDMenvir), 8
- print, 14, 64
  
- print.tdmClass, 18
- print.tdmClass (print.TDMclassifier), 9
- print.TDMclassifier, 9, 20, 21
- print.TDMdata, 10
- print.tdmRegre, 56
- print.tdmRegre (print.TDMregressor), 11
- print.TDMregressor, 11
  
- randomForest, 45
- rCMA, 3, 13
- rSFA, 40, 50
  
- setParams, 4–6, 11
- sfaPBootstrap, 40
- sfaStep, 51
- SPOT, 3
- spot, 13
- spotControl, 5
- spotTuner, 13
  
- tdmBigLoop, 3, 12, 22, 25, 26, 53, 63, 64, 67
- tdmBindResponse, 15
- tdmClass, 7–10
- TDMclassifier, 7–10, 14, 18, 19, 60, 61, 63, 64, 67
- TDMclassifier (tdmClassifyLoop), 18
- tdmClassify, 10, 16, 19–21, 34, 41, 42, 44, 57, 58
- tdmClassifyLoop, 3, 10, 16–18, 18, 21, 39, 42, 53, 56, 58, 60, 61
- tdmClassifySummary, 10, 20, 59
- TDMdata, 6, 7, 10, 13, 52, 53, 63
- TDMdata (tdmReadAndSplit), 52
- tdmDefaultsFill, 4, 13, 21, 25, 26, 28, 43, 63, 66
- tdmDispatchTuner, 12, 14, 22, 62, 64
- TDMenvir, 7–9, 25, 26
- TDMenvir (tdmEnvTMakeNew), 25
- tdmEnvTAddBstRes, 23
- tdmEnvTAddGetters, 24, 25

- tdmEnvTGetOpts, 24
- tdmEnvTLoad, 25
- tdmEnvTMakeNew, 25
- tdmEnvTReport, 26, 27
- tdmEnvTReportSens, 26, 27
- tdmEnvTSetOpts, 27
- tdmEnvTUpdate, 28
- tdmGraAndLogFinalize, 28
- tdmGraAndLogInitialize, 29
- tdmGraphicCloseDev, 29
- tdmGraphicCloseWin, 30
- tdmGraphicInit, 28–30, 30, 31
- tdmGraphicNewWin, 31
- tdmGraphicToTop, 31
- tdmMapDesApply, 32, 32, 33
- tdmMapDesLoad, 32, 32
- tdmModConfmat, 33, 42
- tdmModCreateCVindex, 19, 35, 39, 54, 57
- tdmModSortedRFimport, 16, 17, 36, 36, 38, 40
- tdmModVote2Target, 37
- tdmOptsDefaultsFill, 39, 42, 43
- tdmOptsDefaultsSet, 4, 38, 42
- tdmParaBootstrap, 42, 43
- tdmPreAddMonomials, 44
- tdmPreFindConstVar, 45
- tdmPreGroupLevels, 40, 45
- tdmPreLevel2Target, 46
- tdmPreNARoughfix, 46
- tdmPrePCA.apply, 47, 49
- tdmPrePCA.train, 47, 48
- tdmPreSFA.apply, 49, 51
- tdmPreSFA.train, 49, 50, 50
- TDMR (TDMR-package), 3
- TDMR-package, 3
- tdmRandomSeed, 17, 19, 39, 41, 51, 52
- tdmReadAndSplit, 6, 7, 10, 13, 22, 52, 54, 55, 63
- tdmReadDataset, 39, 52, 53, 53, 54
- tdmReadTaskData, 54
- tdmRegre, 7, 8, 11
- tdmRegress, 11, 20, 41, 42, 55, 57–59
- tdmRegressLoop, 3, 11, 18, 20, 53, 55, 56, 57, 59
- TDMregressor, 7–9, 11, 14, 57, 58, 63, 67
- TDMregressor (tdmRegressLoop), 57
- tdmRegressSummary, 11, 21, 59
- tdmROCR.TDMclassifier, 60, 61
- tdmROCRbase, 34, 60, 61
- tdmStartSpot2, 22
- tdmTuneIt, 3, 62
- twiddle, 60
- unbiasedRun, 6, 8, 12, 14, 22, 54, 62–64, 65