The TDMR Tutorial: Examples for Tuned Data Mining in R

Wolfgang Konen, Patrick Koch,
Cologne University of Applied Sciences

Initial version: June, 2012
Last update: August, 2014

1 Overview

The TDMR framework is written in R with the aim to facilitate the training, tuning and evaluation of data mining (DM) models. It puts special emphasis on tuning these data mining models as well as simultaneously tuning certain preprocessing options.

This document (TDMR-tutorial.pdf)

- describes the TDMR **installation**
- shows **example usages**: how to use TDMR on new data mining tasks
- provides a **FAQ-section** (frequently asked questions)

This document should be read in conjunction with the companion document TDMR-docu.pdf [Konen and Koch(2012a)], which describes more details and software concepts of TDMR.

Both documents are available online as CIOP Reports (PDF, [Konen and Koch(2012a), Konen and Koch(2012b)]).

Both documents concentrate more on the software usage aspects of the TDMR package. For a more scientific discussion of the underlying ideas and the results obtained, the reader is referred to [Kone10a, Kone11b].

2 Installing TDMR

Once you have R (http://www.r-project.org/), > 2.14, up and running, simply install TDMR with

```
install.packages("TDMR");
```

Then, library TDMR is loaded with

```
library(TDMR);

## Loading required package:  testit
## Loading required package:  SPOT
## Loading required package:  rpart
## Loading required package:  emoa
```

3 Lessons

NOTE: Many, but not all TDMR demos and functions will run under RStudio. This is due to some incompatibilities in RStudio's graphic device(s). All demos and functions will however run under RGui.

To start a demo, e.g. `demo/demo00-0classif.r`, type

```
demo("demo00-0classif")
```

or

```
demo("demo00-0classif",ask=F)
```

3.0 Lesson 0: A simple TDMR program

`demo/demo00-0classif.r`
`demo/demo00-1regress.r`

This demo shows the most simple TDMR program. It does not need any external files.

```
##+ --------- demo/demo00-0classif.r ---------
# set all defaults for data mining process:
opts=tdmOptsDefaultsSet()
opts$TST.SEED=5                            # reproducible results
gdObj <- tdmGraAndLogInitialize(opts); # init graphics and log file

data(iris)
response.vars="Species"                    # names, not data (!)
input.vars=setdiff(names(iris),"Species")

result = tdmClassifyLoop(iris,response.vars,input.vars,opts)
```

Here, `tdmOptsDefaultsSet` will construct a default list `opts` with all relevant settings. See TDMR-docu.pdf [Konen and Koch(2012a)], Appendix B, for a complete list of all elements and all defaults for list `opts`. After initializing graphics and log file, the dataset `iris` is loaded and the target (`Species`) as well as the input variables (all other column names from `iris`) are defined.

Now the classification DM task is started with `tdmClassifyLoop`.

Inside `tdmClassifyLoop` the following things happen:

Data partitioning: The dataset will be divided by random sampling in a training set (90%) and validation set (10%), based on `opts$TST.kind="rand", opts$TST.valiFrac=0.1`.

Variable selection: Since you do not specify anything from the `opts$SRF`-block (**s**orted **r**andom **f**orest importance), you use the default SRF variable ranking (`opts$SRF.kind ="xperc"`, `opts$SRF.Xperc=0.95`). This means that the most important columns (containing not less than 95% of the overall importance) will be selected.

Modeling and evaluation: Since you do not specify anything else, function `tdmClassifyLoop` builds an RF (`randomForest`) model (`opts$MOD.method="RF"`) using the training data and evaluates it on training and validation data. It returns an object `result`. The object `result` of class `TDMclassifier` is explained in more detail in Table 3 of TDMR-docu.pdf [Konen and Koch(2012a)].

Repeated runs: Since the default setting `opts$NRUN=2` is used, the whole procedure (random partitioning into training and validation set, RF-based selection of the most important variables, model building, and model evaluation) is repeated 2 times in 2 runs with different random seeds. The different runs are aggregated (usually by averaging).

We now take a look at the output generated by `tdmClassifyLoop`. Since we do not change the default `opts$VERBOSE=2`, TDMR will print a lot of diagnostic output:

```
## default.txt : Stratified random training-validation-index with opts$TST.valiFrac =  10 %
##
## default.txt : Importance check ...
## Clipping sampsize to  135

## Loading required package:  randomForest
## Warning:  package 'randomForest' was built under R version 3.0.3
## randomForest 4.6-7
## Type rfNews() to see new features/changes/bug fixes.

## default.txt : Train RF (importance, sampsize= 135 ) ...
## default.txt : Saving SRF (sorted RF) importance info on opts ...
## Variables sorted by importance (4 ):
## [1] "Petal.Width"  "Petal.Length" "Sepal.Length" "Sepal.Width"
## Dropped columns (0 [=  0.0% of total importance]):
## Proc time:  0.04
## Run  1 / 2 :
## default.txt : Train RF with sampsize = 135 ...
## Proc time:  0.11
## default.txt : Apply RF ...
## Proc time:  0
## default.txt : Calc confusion matrix + gain ...
##
## Training cases ( 135 ):
##             predicted
## actual       setosa versicolor virginica
##   setosa         45          0         0
##   versicolor      0         42         3
##   virginica       0          4        41
## total gain:   128.0 (is  94.815% of max. gain =   135.0)
##
## Validation cases ( 15 ):
##             predicted
## actual       setosa versicolor virginica
##   setosa          5          0         0
##   versicolor      0          5         0
##   virginica       0          1         4
##             setosa versicolor virginica Total
## gain.vector      5          5         4    14
## total gain :    14.0 (is  93.333% of max. gain =    15.0)
##
##   Relative gain on   training set    94.81 %
##   Relative gain on validation set    93.33 %
##
## default.txt : Stratified random training-validation-index with opts$TST.valiFrac =  10 %
##
## default.txt : Importance check ...
## Clipping sampsize to  135
## default.txt : Train RF (importance, sampsize= 135 ) ...
## default.txt : Saving SRF (sorted RF) importance info on opts ...
## Variables sorted by importance (4 ):
## [1] "Petal.Length" "Petal.Width"  "Sepal.Length" "Sepal.Width"
## Dropped columns (1 [=  0.5% of total importance]):
## [1] "Sepal.Width"
## Proc time:  0.02
## Run  2 / 2 :
## default.txt : Train RF with sampsize = 135 ...
## Proc time:  0.06
## default.txt : Apply RF ...
## Proc time:  0.03
## default.txt : Calc confusion matrix + gain ...
##
## Training cases ( 135 ):
##             predicted
## actual       setosa versicolor virginica
##   setosa         45          0         0
##   versicolor      0         42         3
##   virginica       0          4        41
## total gain:   128.0 (is  94.815% of max. gain =   135.0)
##
## Validation cases ( 15 ):
##             predicted
## actual       setosa versicolor virginica
##   setosa          5          0         0
##   versicolor      0          5         0
##   virginica       0          0         5
##             setosa versicolor virginica Total
## gain.vector      5          5         5    15
## total gain :    15.0 (is 100.000% of max. gain =    15.0)
##
##   Relative gain on   training set    94.81 %
##   Relative gain on validation set    100 %
##
##
## Average over all  2  runs:
## cerr$train: (5.18519 +- 0.00000)%
## cerr$vali:  (3.33333 +- 4.71405)%
## gain$train: ( 128.00 +- 0.00)
## gain$vali:  (  14.50 +- 0.71)
## rgain.train:  94.815%
## rgain.vali:   96.667%
```

The first line tells us that TDMR has set aside 10% of the data (15 records in the case of `iris` with 150 records) for validation, the remaining 135 are for training. A random forest is trained to assess the importance of the input variables. We get with

```
[1] "Petal.Width" "Petal.Length" "Sepal.Length" "Sepal.Width"
```

the variables sorted by decreasing importance. It depends on the importance of the least important variable (here: `Sepal.Width`) whether it will be dropped or not. In the first run it is not dropped, because its importance is above the threshold $1-0.95 = 5\%$. In the second run it is dropped, because due to statistical fluctuations now its importance is with 0.5% below the threshold of 5%.

In the next step the DM model (here: RF) is trained with the selected variables and then the trained model is applied to the training data and to the validation data. In each case the confusion matrix (actual vs. predicted) is shown. In the case of RF, the prediction on the training data is the OOB prediction. The `total gain` reported is the sum of the element-wise product „gain matrix × confusion matrix“ where the gain matrix denotes for every classification outcome „actual vs. predicted“ the associated gain.[1] If nothing else is

[1]In this toy problem, the gain on the validation set is statistically not very meaningful since the validation

defined, the gain matrix is the identity matrix. The relative gain is defined as

$$\texttt{rgain} = \frac{\sum_{ij} G_{ij} C_{ij}}{\sum_{ij} G_{ij} C_{ij}^{(ideal)}} \qquad \text{with} \qquad G = \text{gain matrix} \quad \text{and} \quad C = \text{confusion matrix}$$

where $C^{(ideal)}$ is the perfect confusion matrix (all records appear on the main diagonal).

Finally, all runs (2 in this example) are averaged and the average classification error `cerr`, the average gain, and the average relative gain `rgain` are reported both for the training and the validation set.

A similar small sample program exists for regression (`demo/demo00-1regress.r`).

3.1 Lesson 1: DM on task SONAR

`demo/demo01-1sonar.r`
`demo/demo01-2cpu.r`

This lesson demonstrates the usage of TDMR for a somewhat bigger DM task: data are read from file and the information for controlling TDMR is distributed over several files. This may look complicated at first sight, but it is useful for two reasons:

1. As a preparation for the tuning process in the further lessons: It is very useful if we can package the whole data mining process (from training-validation-data generation over model building up to model evaluation) into one function or file. It will be easily callable by the tuner.
2. For conducting slightly different variants, runs or experiments, it is useful to package the parameter setting part in one (or several) files as well.

In this lesson we will look at four relevant files:

1. `sonar_00.apd` (the parameter settings)
2. `main_sonar.r` (the DM function `main_sonar`)
3. `start_sonar.r`
4. `demo01-1sonar.r`

Suppose that you have a dataset and want to build a DM model for it. To be concrete, we consider the classification dataset SONAR[2] with the data file `sonar.txt`.

If you want to build a DM classification model with TDMR, you need to provide two files, `sonar_00.apd` and `main_sonar.r`.[3] The first file, `sonar_00.apd` (`.apd` = algorithmic problem

set has only 15 records.

[2]see UCI repository or package mlbench for further info on SONAR)

[3]Templates for `sonar_00.apd` and `main_sonar.r` are available from `<inst>/demo02sonar` where `<inst>` refers to the installation directory of package TDMR as returned by `find.package("TDMR")`.

design), is already in preparation for later tuning (see Lesson02 and Lesson03), it defines in list `opts` all relevant settings for the DM model building process. The second file, `main_sonar.r`, contains this DM model building process. It gets with list `opts` the settings and returns in list `result` the evaluation of the DM model. The list `result` is either inspected by the user or by the tuning process.

```
## sonar_00.apd
##
## set the basic elements of list opts for task sonar. See ?tdmOptsDefaultsSet
## for a complete list of all default settings and many explanatory comments
      opts = tdmOptsDefaultsSet();
      opts$dir.data <- "data/";
      opts$filename = "sonar.txt"
      opts$READ.CMD = "readCmdSonar(filename,opts)"
      opts$data.title <- "Sonar Data"
```

Here, `tdmOptsDefaultsSet()` will construct a default list `opts` with all relevant settings. See TDMR-docu.pdf [Konen and Koch(2012a)], Appendix B, for a complete list of all elements and all defaults for list `opts`. You need to specify only those things which differ from `tdmOptsDefaultsSet()`: in this case most importantly the filename and directory of the SONAR dataset and a string `opts$READ.CMD` containing the data-reading command.

The file `main_sonar.r` contains two functions `main_sonar` and `readCmdSonar`:

```
main_sonar <- function(opts=NULL, dset=NULL, tset=NULL) {
  if (is.null(opts)) source("sonar_00.apd", local=TRUE);
  opts <- tdmOptsDefaultsSet(opts);    # fill in all opts params not yet set

  gdObj<-tdmGraAndLogInitialize(opts); # init graphics and log file

  ########  PART 1: READ DATA    ##########################
  if (is.null(dset)) {
      cat1(opts,opts$filename,": Read data ...\n")
      dset <- tdmReadData(opts);
  }
  names(dset)[61] <- "Class"  # 60 columns V1,...,V60 with input data, one
                              # response column "Class" with levels ["M"|"R"]

  response.vars <- "Class"             # which variable(s) are target

  # which variables are input variables (in this case all others):
  input.vars <- setdiff(names(dset), c(response.variable))

  ########  PART 2: Model building and evaluation #########
  result <- tdmClassifyLoop(dset,response.vars,input.vars,opts,tset);
  # print summary output and attach certain columns
  # (here: y, sd.y, dset) to list result:
  result <- tdmClassifySummary(result,opts,dset);

  tdmGraAndLogFinalize(opts,gdObj);  # close graphics and log file

  result;
}
```

```
readCmdSonar <- function(filename,opts) {
  read.csv2(file=paste(opts$dir.data, filename, sep=""),
            dec=".", sep=",", nrow=opts$READ.NROW, header=FALSE);
}
```

To start the whole procedure, there is a small starter file `start_sonar.r`:

```
source("main_sonar.r");
result <- main_sonar(opts);
```

This file is invoked by `demo01-1sonar.r`:

```
##* --------- demo/demo01-1sonar.r ---------
path <- paste(find.package("TDMR"), "demo02sonar",sep="/");
source(paste(path,"sonar_00.apd",sep="/"),local=TRUE);   # set opts
source(paste(path,"start_sonar.r",sep="/"),chdir=TRUE);
```

The reason why we have the file chain

$$\texttt{demo01-1sonar.r} \xrightarrow{\texttt{source}} \texttt{start\_sonar.r} \xrightarrow{\texttt{source}} \texttt{main\_sonar.r}$$

is the following: `main_sonar` may need to perform certain file I/O in the directory `path`. Sourcing `start_sonar.r` with `source(...,chdir=TRUE)` tells R that it changes to the directory `path` prior to sourcing (and automatically returns to the actual working directory at the end of sourcing[4]).

Note that the special path with `find.package("TDMR")` and the distinction between `start_sonar.r` and `demo01-1.sonar.r` is only needed for the TDMR-package demo which requires the demo R-script and the data directory to be in different (and TDMR-package-specific) directories. – If you write your own application, you can have `main_sonar.r` and `sonar_00.apd` in the same directory `myDir` at any place on your computer. The data file `sonar.txt` should be in the subdirectory `myDir/data`. Then you only need one starter script `start_myApp.r` in `myDir` which simply reads like this:

[4]Even in the case of an error inside `start_sonar.r` R will correctly return to the actual working directory.

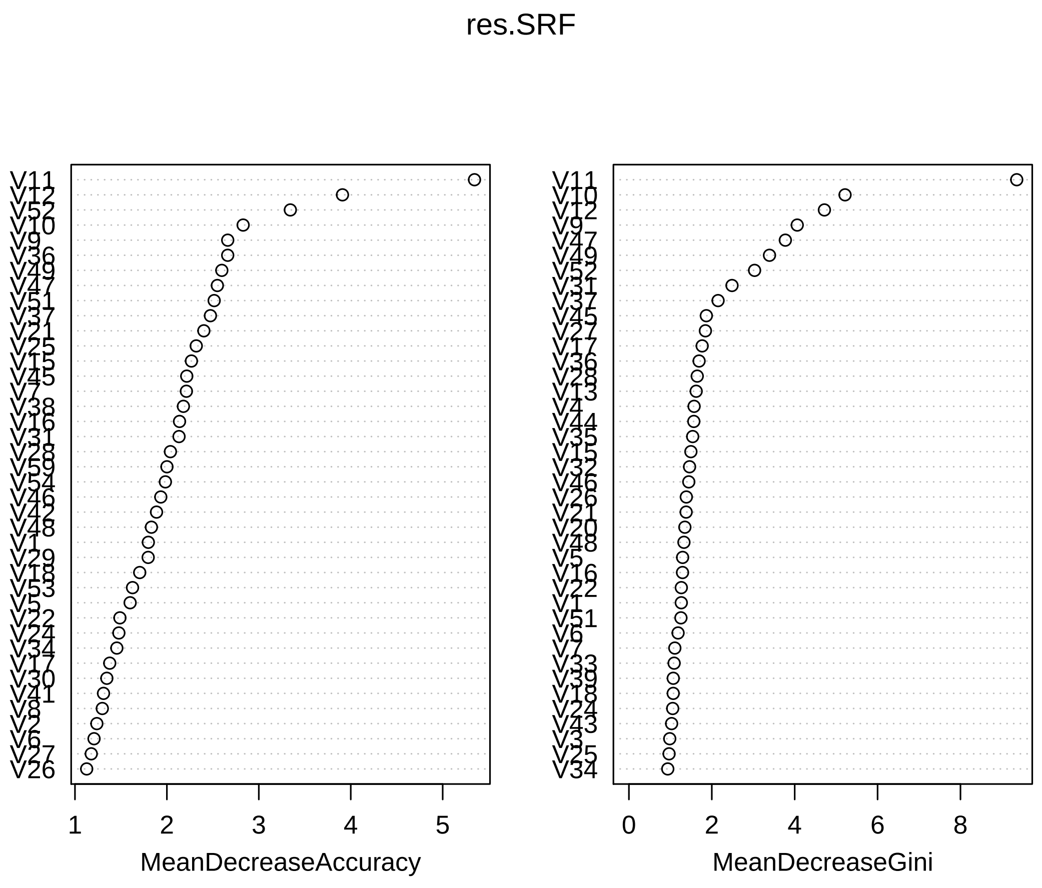


Figure 1: The first plot from demo01-1sonar.r shows the variable importance.

```
source("sonar_00.apd",local=TRUE)
source("main_sonar.r");
result <- main_sonar(opts);
```

We now take a closer look at function `main_sonar`.

Data reading: Function `main_sonar` is called here with argument `opts` (built via `sonar_00.apd`). Part 1 reads the dataset `dset` from file, and defines the input variables and the target variable `response.vars`.

As a side remark: From where is the dataset `dset` read? - TDMR searches in the working directory the file `opts$dir.data/opts$filename` and reads it with command `readCmdSonar`. More precisely: The setting

```
opts$READ.CMD = "readCmdSonar(filename,opts)"
```

tells TDMR that TDMR's function `tdmReadData` should invoke `readCmdSonar` and pass the value of `opts$dir.data/opts$filename` to `readCmdSonar`'s argument `filename`. Any other user-defined function can be supplied in `opts$READ.CMD` as well, the only rules are

- it has to return a data frame (which becomes TDMR's variable `dset`)
- the string `opts$READ.CMD` has to contain the argument `filename`.

Data selection, modeling and evaluation: Part 2 of function `main_sonar` starts the DM model building process with

```
result <- tdmClassifyLoop(dset,response.vars,input.vars,opts,tset);
```

See Lesson 0 in Sec. 3.0 for an in-depth description of what is happening inside `tdmClassifyLoop`. The principle is the same, it is now only applied to another data set `sonar.txt`.

3.2 Lesson 2: SPOT tuning on task SONAR

`demo/demo02sonar.r`

If you want to do a SPOT tuning [Bartz-Beielstein(2010)] on task SONAR, you should follow the steps described in TDMR Workflow, Level 2 and create in addition to `main_sonar.r` from Lesson01 the three small files `sonar_01.conf`, `sonar_01.apd` and `sonar_01.roi`. The content of these files may look for example like this:

sonar_01.conf

```
alg.func = "tdmStartSpot"
alg.resultColumn = "Y"
alg.seed = 1235

io.apdFileName = "sonar_01.apd"
io.roiFileName = "sonar_01.roi"
spot.seed = 120 # 125
io.verbosity = 3;
auto.loop.steps = 50;   # number of spot metamodels to be generated
auto.loop.nevals = 50;  # concurrently, max number of algo evaluations

init.design.func = "spotCreateDesignLhd";
init.design.size = 10;    # number of initial design points
init.design.repeats = 1;  # number of initial repeats

seq.merge.func <- mean;
seq.design.size = 100;
seq.design.retries = 15;
seq.design.maxRepeats = 2;
seq.design.oldBest.size <- 1;
seq.design.new.size <- 3;

seq.predictionModel.func = "spotPredictRandomForest";

report.func = "spotReportSens"
```

sonar_01.apd

```
  opts = tdmOptsDefaultsSet();
  opts$dir.data <- "data/";
  opts$filename = "sonar.txt"
  opts$READ.CMD = "readCmdSonar(filename,opts)"    # defined in main_sonar.r
  opts$data.title <- "Sonar Data"

  opts$RF.mtry = 4
  opts$NRUN =  1          # how many runs with different train & vali samples
                          # - or - how many CV-runs, if TST.kind="cv"
  opts$GD.DEVICE="non"    # e.g. ["pdf"|"win"|"non"]
  opts$GD.RESTART=F;
  opts$VERBOSE = opts$SRF.verbose = 0;
  #opts£logFile=FALSE      # no logfile (needed for Sweave/.Rnw only)
```

sonar_01.roi

```
name low high type
CUTOFF1 0.1 0.80 FLOAT
CLASSWT2 5 15 FLOAT
XPERC 0.90 1.00 FLOAT
```

The three parameters `CUTOFF1`, `CLASSWT2` and `XPERC` are tuned within the borders specified by `sonar_01.roi`. Usually you should set `opts$GRAPHDEV="non"` and `opts$GD.RESTART=F` to avoid any graphic output and any graphics device closing from `main_sonar.r`, so that you get only the graphics made by SPOT.

To start the whole procedure, there is a small starter file `start_bigLoop.r`:

```
envT <- tdmEnvTMakeNew(tdm);
envT <- tdmBigLoop(envT,spotStep);
```

This file is invoked by `demo02sonar.r`:

```
#*# --------- demo/demo02sonar.r ---------
path <- paste(find.package("TDMR"), "demo02sonar",sep="/");
tdm=list(mainFile="main_sonar.r"
        ,runList="sonar_01.conf"
        );
spotStep = "auto";
source(paste(path,tdm$mainFile,sep="/"));
source(paste(path,"start_bigLoop.r",sep="/"),chdir=TRUE);
```

The reason why we have the file chain

$$\texttt{demo02sonar.r} \xrightarrow{\texttt{source}} \texttt{start\_bigLoop.r} \xrightarrow{\texttt{source}} \texttt{tdmEnvTMakeNew}$$

is the same as in Lesson 1: `tdmEnvTMakeNew` may need to perform certain file I/O in the directory `path`. Sourcing `start_bigLoop.r` with `source(...,chdir=TRUE)` tells R that it changes to the directory `path` prior to sourcing (and automatically returns to the actual working directory at the end of sourcing[5]).

Again, as in Lesson 1, the distinction between `start_bigLoop.r` and `demo02sonar.r` is only needed for the TDMR-package demo. If you write your own application, you can have `main_sonar.r` together with the `.apd`, `.roi` and `.conf` files in the same directory `myDir` at any place on your computer. The data file `sonar.txt` should be in the subdirectory `myDir/data`. Then you only need one starter script `start_myBigLoop.r` in `myDir` which simply reads like this:

```
tdm=list(mainFile="main_sonar.r"
        ,runList="sonar_01.conf"
        );
source(paste(path,tdm$mainFile,sep="/"));
envT <- tdmEnvTMakeNew(tdm);
envT <- tdmBigLoop(envT,"auto");
```

3.3 Lesson 3: „The Big Loop“ on task SONAR

`demo/demo03sonar.r`
`demo/demo03sonar_A.r`
`demo/demo03sonar_B.r`
`demo/demo03newdata.r`

To start „The Big Loop“, you configure a file similar to `demo/demo03sonar.r`:

```
#*# --------- demo/demo03sonar.r ---------
path <- paste(find.package("TDMR"), "demo02sonar",sep="/");
tdm <- list( mainFile="main_sonar.r"
           , runList = c("sonar_04.conf","sonar_06.conf")
           , umode="CV"          # { "CV" | "RSUB" | "TST" | "SP_T" }
```

[5] Even in the case of an error inside `start_bigLoop.r` R will correctly return to the actual working directory.

```
            , tuneMethod = c("lhd")
            , filenameEnvT="demo03.RData"   # save file envT (in dir 'path')
            , nrun=3, nfold=2     # repeats and CV-folds for the unbiased runs
            , nExperim=1
            , parallelCPUs=1
            , optsVerbosity = 3   # the verbosity for the unbiased runs
            );
spotStep = "auto";
source(paste(path,tdm$mainFile,sep="/"));
source(paste(path,"start_bigLoop.r",sep="/"),chdir=TRUE,local=TRUE);
```

This is very much the same as in Lesson 2, we reuse the small starter file `start_bigLoop.r` from there. The only difference is that now **multiple** tuning runs can be performed with respect to the following three dimensions:

- configuration files (elements of `tdm$runList`)
- tuners (elements of `tdm$tuneMethod`)
- repeated experiments with different random seeds (number `tdm$nExperim`).

The function `tdmBigLoop` realizes a triple `for`-loop over these dimensions. With k =`length(runList)`, m =`length(tuneMethod)`, and n =`nExperim` we have in total kmn tuning runs.

Here, the script `demo03sonar.r` will trigger the following sequence of experiments:

- `sonar_04.conf` is started with tuner `lhd`
- `sonar_06.conf` is started with tuner `lhd`.

This sequence of 2 tuning experiments is repeated `nExperim=1` time. The corresponding 2 result lines are written to data frame `envT$theFinals`:

```
print(envT$theFinals);

##        CONF TUNER NEXP CUTOFF1 CLASSWT2  XPERC NRUN NEVAL RGain.bst
## 1 sonar_04   lhd    1 0.09095    5.537 0.6545    3    10     91.67
## 2 sonar_06   lhd    1 0.24287   12.137 0.5520    3    10     97.78
##   RGain.avg RGain.OOB sdR.OOB RGain.CV sdR.CV Time.TST Time.TRN
## 1     77.11     87.56   5.005    87.67  1.528     0.44     1.30
## 2     90.00     95.97   2.010    94.00  2.000     0.44     1.28
```

Here `CUTOFF1`, `CLASSWT2`, and `XPERC` are the tuning parameters, the other columns of the data frame are defined in Table 2 of TDMR-docu.pdf [Konen and Koch(2012a)]. In the case of the example above, the tuning process had a budget of `NEVAL=10` model trainings, resulting in a best solution with class accuracy `RGain.bst` (in %). The average class accuracy (mean w.r.t. all design points) during tuning is `RGain.avg`. When the tuning is finished, the best solution is taken and `NRUN=3` unbiased evaluation runs are done with the parameters of the best solution. Since the classification model in this example is RF (Random Forest), an OOB-error with mean `RGain.OOB` from the 3 trainings is returned. Additionally, `NRUN=3` trainings are done with cross validation (CV) with new randomly created folds in each run, resulting in an average class accuracy `RGain.CV`. For each measure `RGain.*` there is also an accompanying column `sdr.*` giving the standard deviation with respect to the `NRUN` runs.

Tuning runs are rather short, to make this 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.

We now add an extra feature to this demo lesson: Suppose you have a large dataset and you want to do quick tuning runs. To reduce the tuning time (of course at the price of a somewhat reduced tuning quality) you may specify the parameter `opts$READ.NROW` to a value smaller than the size of the dataset. Then only this number of records is read and used for training and validation during tuning. After tuning has finished, you may want to use the best parameters found by tuning and to perform a high-quality training and evaluation on the full dataset to assess the real strength of the tuning result.

In our demo lesson we have specified in `sonar_06.apd` the line

```
opts$READ.NROW = 100
```

For the SONAR dataset containing only 208 records, the reduction is of course quite meaningless, it serves only as a demonstration. But for large datasets with e.g. 100 000 records, the time reduction can be substantial. The tuning results were saved in `demo03.RData`. We load this file, re-source `sonar_06.apd` and then set `opts$READ.NROW=-1`. This means that we now read **all** data with `tdmSplitTestData` and enter `tdmBigLoop` with this dataset `dataObj` and with `spotStep="rep"` indicating that we grab the best tuning result and perform training and evaluation on the new dataset:

```
#*# --------- demo/demo03newdata.r ---------
path <- paste(find.package("TDMR"), "demo02sonar",sep="/");
oldwd <- getwd(); setwd(path);
envT <- tdmEnvTLoad("demo03.RData");
source(envT$tdm$mainFile);
source("sonar_06.apd")      # opts
opts$READ.NROW=-1;
envT$tdm$optsVerbosity=3;
dataObj <- tdmSplitTestData(opts,envT$tdm);
envT <- tdmBigLoop(envT,"rep",dataObj);
setwd(oldwd);
```

Note that the dataset `dataObj`, when specified in `tdmBigLoop`, is used for **every** run (every CONF file) in the big loop.[6]

[6]If `dataObj` were not specified in the call to `tdmBigLoop`, each CONF file would construct its own `dataObj` inside the loop. Then, however, with the very same parameters as used during tuning.

The results of the new unbiased evaluation runs are again recorded in `envT$theFinals`:

```
print(envT$theFinals);

##       CONF TUNER NEXP CUTOFF1 CLASSWT2  XPERC NRUN NEVAL RGain.bst
## 1 sonar_04   lhd    1 0.09095    5.537 0.6545    3    10     91.67
## 2 sonar_06   lhd    1 0.24287   12.137 0.5520    3    10     97.78
##   RGain.avg RGain.OOB sdR.OOB RGain.CV sdR.CV Time.TST Time.TRN
## 1     77.11     87.56   5.005    60.26  2.272     1.14     1.29
## 2     90.00     95.97   2.010    70.67  3.002     1.10     1.28
```

3.4 Lesson 4: Regression Big Loop

`demo/demo04cpu.r`

The same as Lesson 3, but applied to a regression task (dataset CPU).

3.5 Lesson 5: Interactive Visualization

`demo/demo05visMeta.r`

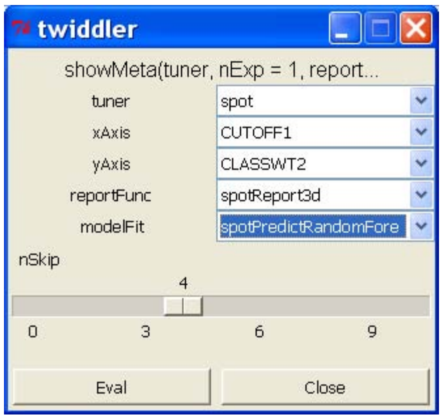


Figure 2: The user interface in `tdmPlotResMeta`. The user may select the tuner, the design variables to show on x- and y-axis, the display function (`spotReport3d` or `spotReportContour`) and the metamodel function (`modelFit`). Two optional sliders are `nExper` and `nSkip` (see text).

Once a Lesson-3 experiment is completed, the return value `envT` from `tdmBigLoop()` contains the result of such an experiment and may be visually inspected. Alternatively, `envT` may be loaded from an appropriate `.RData` file. The call

```
tdmPlotResMeta(envT);
```

allows to visually inspect all RES data frames contained in `envT`.

The user interface is shown and explained in Fig. 2. An additional combo box `confFile` appears only, if `envT$runList` has more than one element. An additional slider `nExper` appears only, if `envT$tdm$nExperim>1`.

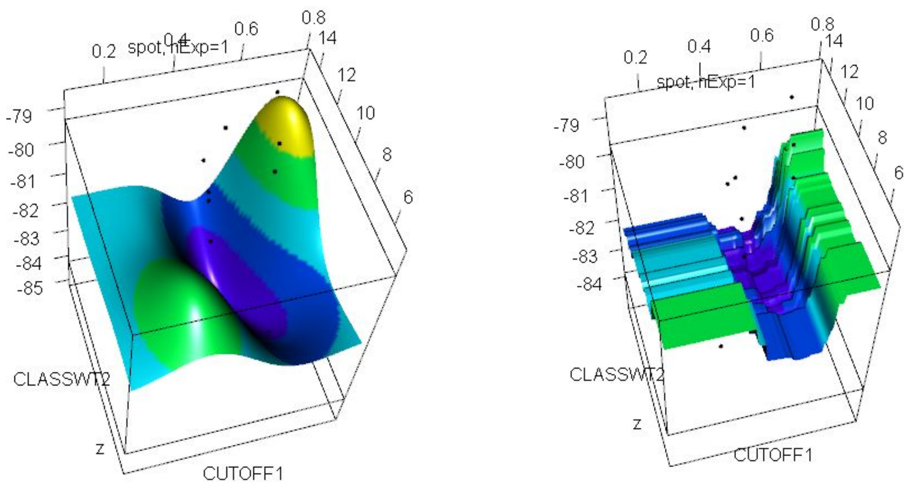


Figure 3: Two example outputs from `tdmPlotResMeta` with `reportFunc=spotReport3d`. Left: `modelFit = spotPredictGausspr`, right: `= spotPredictRandomForest`.

The user selects with `tuner`, `confFile` and `nExper` a certain RES data frame from `envT`. This data frame contains a collection of function evaluations for certain design points selected by the tuner. With one of the metamodel construction functions (see package SPOT for further details)

- spotPredictGausspr
- spotPredictRandomForest
- spotPredictMlegp

a metamodel is fitted to the RES data frame and the result is shown as shaded surface in the plot. The RES data points are shown as black points in Fig. 3. Since certain "bad" RES point may dominate the plot as outliers and hinder the user to inspect the region near the optimum, there are two options to suppress "bad" points:

1. If the slider `nSkip` has a value > 0, then the `nSkip` RES data points with the worst y-value are discarded.

2. If the checkbox ”Skip incomplete CONFIGs” is activated, then design points belonging to a configuration which was not evaluated `maxRepeats` times are discarded (relevant for SPOT only).

Note that both options will reduce the number of RES data points. This will also affect the metamodel fit, so use both options with care, if the number of RES data points is small.

The plots created with `spotReport3d` make use of the rgl-package. They can be interactively manipulated with the mouse. They can be selected and saved as PNG images with commands like

```
rgl.set(7);
rgl.snapshot("myFile.png");
```

A complete demo example is invoked with:

```
demo(demo05visMeta);
```

3.6 Lesson 6: Performance Measure Plots

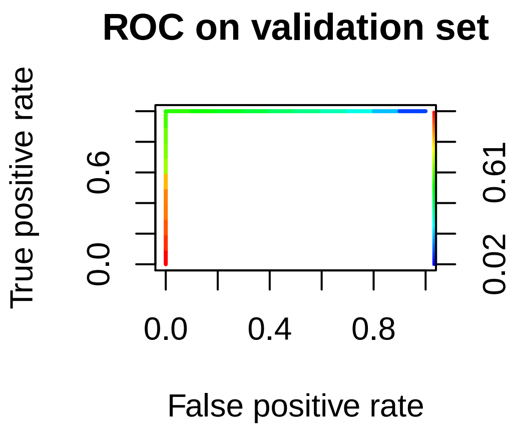
`demo/demo06ROCR.r`

With the help of package ROCR [Sing et al.(2005)Sing, Sander, Beerenwinkel, and Lengauer], several area performance measures can be used for binary classification. The file `demo/demo06ROCR.r` shows an example:

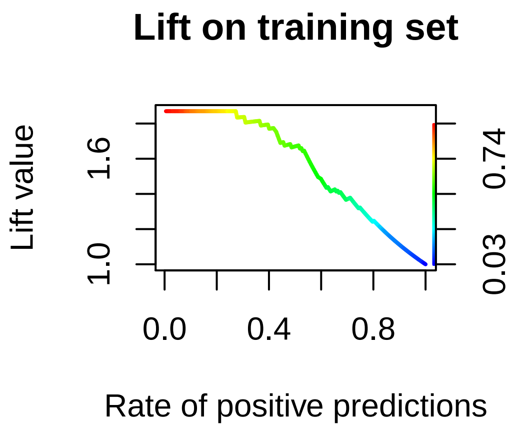
```
  opts = tdmOptsDefaultsSet();
  opts$filename = "sonar.txt"
  opts$READ.CMD = "readCmdSonar(filename,opts)"    # defined in main_sonar.r
  opts$data.title <- "Sonar Data";
  opts$rgain.type <- "arROC";
  path <- paste(find.package("TDMR"), "demo02sonar",sep="/");
  source(paste(path,"start_sonar.r",sep="/"),chdir=TRUE);

## Loading required package:  ROCR
## Loading required package:  gplots
## KernSmooth 2.23 loaded
## Copyright M. P. Wand 1997-2009
##
## Attaching package:  'gplots'
##
## The following object is masked from 'package:stats':
##
##    lowess
```

As explained in Lesson 1 in more detail, the file `start_sonar.r` contains the line



(a) ROC curve



(b) Lift chart

Figure 4: (a) ROC curve on validation set with `tdmROCRbase(result)`; (b) Lift chart on training set with `tdmROCRbase(...,typ="lift")`. The bar on the right side shows a color coding of the cutoff parameter.

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

Once the variable `result` contains an object of class `TDMclassifier`, we can infer from it with `tdmROCRbase` the area under the ROC curve and – as a side effect – plot the ROC curve (Fig. 4(a)).

```
  cat("Area under ROC-curve for validation data set: ",
      tdmROCRbase(result),"\n");     # side effect: plot ROC-curve

## Area under ROC-curve for validation data set:  1
```

Equally well we can infer with `typ="lift"` the area under the lift curve and plot a lift chart (Fig. 4(b))

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

## Area under lift curve for  training data set:  0.5649
```

Once the variable `result` contains an object of class `TDMclassifier`, it is also possible to inspect such an object interactively with the following command:

```
tdmROCR(result);
```

A `twiddler` interface for object `result` shows up (Fig. 5) and allows to select between

- different performance measure plots (ROC-, lift- or precision-recall-chart)
- different data sets (training set or validation set)
- different runs stored in object `result`.

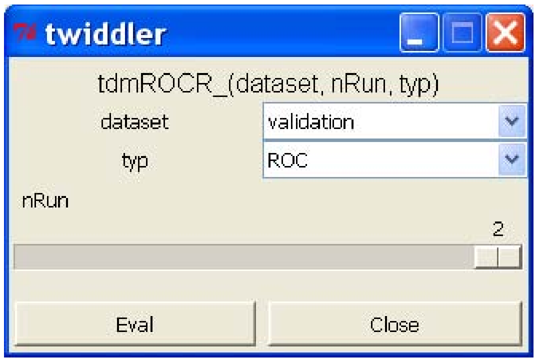

tdmROCR_(dataset, nRun, typ)
dataset: validation
typ: ROC
nRun: 2

Figure 5: Twiddler interface for `tdmROCR(result)`. The user may select the dataset (training or validation), the type of plot (ROC, lift, or precision-recall) and the number of the run (only if `Opts(result)$NRUN>1`).

3.7 Lesson 7: Tuner CMA-ES (rCMA)

`demo/demo07cma_j.r`

This demo shows for tuner `cma_j` (CMA-ES, [?], Java version interfaced to R via package rCMA) a complete tuned data mining process (TDMR, level 3). Other settings are the same as in `demo03sonar.r`, except that we use `sonar_03.conf` as configuration file.

3.8 Lesson 8: Parallel TDMR

`demo/demo08parallel.r`

This demo does the same as `demo03sonar.r`, but it runs 4 experiments on 4 parallel cores (if your environment supports parallel clusters with the R core-package parallel).

4 Frequently Asked Questions (FAQ)

4.1 I have already obtained a best tuning solution for some data set. How can I rerun and test it on the same / other data?

Rerun your Lesson-3 script with `spotStep ="rep"`, this will re-use the current best solution in environment `envT`.

Or use the following code snippet:

```
path <- paste(find.package("TDMR"), "demo02sonar",sep="/");
source(paste(path,"start_rerun.r",sep="/"),chdir=TRUE);
```

The file `start_rerun.r` contains:

```
envT = tdmEnvTLoad("demoSonar.RData");     # load envT
source("main_sonar.r");
envT$tdm$nrun=2;        # =0: no unbiasedRun
finals = tdmEnvTSensi(envT,1);
if (!is.null(finals)) print(finals);
```

Line 1 loads a previously constructed `envT` from an `.RData` file.

Line 4 would make solely the sensitivity plot (w/o unbiased runs), if `envT$tdm$nrun` were 0. But here we set `envT$tdm$nrun=2`, i.e. two unbiased runs with the best tuning solution contained in `envT` are done with the usual test data set.

4.2 How can I make with a trained model new predictions?

Run your Lesson-3 script or Lesson-4 script to produce an environment `envT`, which is an object of class `TDMenvir`. There is an element `lastModel` defined in `envT` which contains the model trained on the best tuning solution during the last unbiased run. TDMR defines a function `predict.TDMenvir` , which makes it easy to do new predictions:

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

Remarks:

- If the new data contain factor variables (e.g. `vendor` in case of CPU data), it is necessary that `levels(newdata$vendor)` is the same as during training. Therefore we read in the above code snippet first all CPU-data and then shorten them to the first 15 records.
- If `envT` is saved to `.RData` file, normally `lastModel` will be set to NULL (smaller `.RData` files). If you want to do predictions, you need to save `lastModel`: to achieve this, set `tdm$U.saveModel=TRUE` prior to running `tdmBigLoop`.
- See also the example in `demo/demo04cpu.r` and in `predict.TDMenvir`.

4.3 How can I add a new tuning parameter to TDMR?

- As a user: Add a new line to `userMapDesign.csv` in directory `tdm$path`.[7] Suppose you want to tune the variable `opts$SRF.samp`: add to file `userMapDesign.csv` a line

  ```
  SRF.SAMP;    opts$SRF.samp;   0
  ```

  This specifies that whenever `SRF.SAMP` appears in a `.roi` file in directory `tdm$path`, the tuner will tune this variable. TDMR maps `SRF.SAMP` to `opts$SRF.SAMP`.

- As a developer: Add similarly a new line to `tdmMapDesign.csv`. This means that the mapping is available for all tasks, not only for those in the current `tdm$path`.

- Optional, as a developer: For a new variable `opts$Z`, add a default-value-line to `tdmOptsDefaultsSet()`. Then all existing and further tasks have this default setting for `opts$Z`.

4.4 How can I add a new machine learning algorithm to TDMR?

See section „How to integrate new machine learning algorithms“ in TDMR-docu.pdf.

4.5 How can I add a new tuning algorithm to TDMR?

See section „How to integrate new tuners“ in TDMR-docu.pdf.

4.6 How can it happen that some variables have an importance that is exactly zero?

Well, the importance for variables with low importance can be zero or even slightly negative (as a consequence of some statistical fluctuations). All those zero or negative importance values will be clipped to zero, therefore a variable with apparently exactly zero importance can happen more frequently than expected.

References

[Bartz-Beielstein(2010)] Thomas Bartz-Beielstein. SPOT: An R package for automatic and interactive tuning of optimization algorithms by sequential parameter optimization. arXiv.org e-Print archive, `http://arxiv.org/abs/1006.4645`, June 2010.

[7] If such a file does not exist yet, the user has to create it with a first line

```
roiValue; optsValue; isInt
```

[Konen and Koch(2012a)] W. Konen and P. Koch. The TDMR Package: Tuned Data Mining in R. Technical Report 02/2012, Research Center CIOP (Computational Intelligence, Optimization and Data Mining), Cologne University of Applied Science, Faculty of Computer Science and Engineering Science, 2012a. URL `http://maanvs03.gm.fh-koeln.de/webpub/CIOPReports.d/Kone12a.d/Kone12a.pdf`.

[Konen and Koch(2012b)] W. Konen and P. Koch. The TDMR Tutorial: Examples for Tuned Data Mining in R. Technical Report 03/2012, Research Center CIOP (Computational Intelligence, Optimization and Data Mining), Cologne University of Applied Science, Faculty of Computer Science and Engineering Science, 2012b. URL `http://maanvs03.gm.fh-koeln.de/webpub/CIOPReports.d/Kone12b.d/Kone12b.pdf`.

[Sing et al.(2005)Sing, Sander, Beerenwinkel, and Lengauer] T. Sing, O. Sander, N. Beerenwinkel, and T. Lengauer. ROCR: visualizing classifier performance in R. *Bioinformatics*, 21(20):3940–3941, 2005. URL `http://rocr.bioinf.mpi-sb.mpg.de/`.