Package ‘rminer’

February 20, 2015

Type Package

Title Data Mining Classification and Regression Methods

Version 1.4

Date 2014-11-06

Author Paulo Cortez

Maintainer Paulo Cortez <pcortez@dsi.uminho.pt>

Description Facilitates the use of data mining algorithms in classification and regression (including time series forecasting) tasks by presenting a short and coherent set of functions. Versions: 1.4 - new classification and regression models/algorithms, with a total of 14 classification and 15 regression methods, including: Decision Trees, Neural Networks, Support Vector Machines, Random Forests, Bagging and Boosting; 1.3 and 1.3.1 - new classification and regression metrics (improved mmetric function); 1.2 - new input importance methods (improved Importance function); 1.0 - first version.

License GPL (>= 2)

URL http://cran.r-project.org/web/packages/rminer/

http://www3.dsi.uminho.pt/pcortez/rminer.html

LazyLoad yes

Depends kknn

Imports methods, plotrix, lattice, nnet, pls, MASS, mda, rpart, randomForest, adabag, party, Cubist, kernlab, e1071

NeedsCompilation no

Repository CRAN

Date/Publication 2014-11-07 14:06:09

R topics documented:

CasesSeries *Create a training set (data.frame) from a time series using a sliding window.*

Description

Create a training set (data.frame) from a time series using a sliding window.

Usage

```
CasesSeries(t, W, start = 1, end = length(t))
```

Arguments

`t`	a time series (numeric vector).
`W`	a sliding window (with time lags, numeric vector).
`start`	starting period.
`end`	ending period.

Details

Check reference for details.

Value

Returns a data.frame, where `y` is the output target and the inputs are the time lags.

Author(s)

Paulo Cortez `http://www3.dsi.uminho.pt/pcortez`

References

- To check for more details:
  P. Cortez.
  Sensitivity Analysis for Time Lag Selection to Forecast Seasonal Time Series using Neural Networks and Support Vector Machines.
  In Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN 2010), pp. 3694-3701, Barcelona, Spain, July, 2010. IEEE Computer Society, ISBN: 978-1-4244-6917-8 (DVD edition).
  http://dx.doi.org/10.1109/IJCNN.2010.5596890

See Also

`fit`, `lforecast`, `predict.fit`.

Examples

```
t=1:20
d=CasesSeries(1:10,c(1,3,4))
print(d)
d=CasesSeries(1:10,c(1,2,3))
print(d)
```

---

crossvaldata *Computes k-fold cross validation for rminer models.*

Description

Computes k-fold cross validation for rminer models.

Usage

```
crossvaldata(x, data, theta.fit, theta.predict, ngroup = 10,
             mode = "stratified", seed = NULL, model, task, feature = "none",
             ...)
```

Arguments

`x`	See `fit` for details.
`data`	See `fit` for details.
`theta.fit`	fitting function
`theta.predict`	prediction function
`ngroup`	number of folds
`mode`	Possibilities are: "stratified", "random" or "order" (see `holdout` for details).
`seed`	if `NULL` then a random seed is used; else a fixed seed is adopted (will return always the same result for the same seed).

`model`	See `fit` for details.
`task`	See `fit` for details.
`feature`	See `fit` for details.
`...`	Additional parameters sent to `theta.fit` or `theta.predic` (e.g. `search`)

Details

Standard k-fold cross-validation adopted for rminer models. By default, for classification tasks ("class" or "prob") a stratified sampling is used (the class distributions are identical for each fold), unless `mode` is set to `random` or `order` (see `holdout` for details).

Value

Returns a list with:

- $cv.fit – all predictions (factor if `task="class"`, matrix if `task="prob"` or numeric if `task="reg"`);
- $mpar – matrix with the mpar for each fold;
- $attributes – the selected attributes for each fold if a feature selection algorithm was adopted;
- $ngroup – the number of folds;
- $leave.out – the computed size for each fold (`=nrow(data)/ngroup`);
- $groups – vector list with the indexes of each group;
- $call – the call of this function;

Note

A better control (e.g. use of several Runs) is achieved using the simpler `mining` function.

Author(s)

This function was adapted by Paulo Cortez from the `crossval` function of the bootstrap library (S original by R. Tibshirani and R port by F. Leisch).

References

Check the `crossval` function of the bootstrap library.

See Also

`holdout`, `fit`, `mining` and `predict.fit`.

Examples

```
data(iris)
# 3-fold cross validation using fit and predict
# the control argument is sent to rpart function
# rpart.control() is from the rpart package
M=crossvaldata(Species~.,iris,fit,predict,ngroup=3,seed=12345,model="rpart",
               task="prob", control = rpart::rpart.control(cp=0.05))
```

```
print("cross validation object:")
print(M)
C=mmetric(iris$Species,M$cv.fit,metric="CONF")
print("confusion matrix:")
print(C)
```

delevels *Reduce (delete) or replace levels from a factor variable (useful for preprocessing datasets).*

Description

Reduce (delete) or replace levels from a factor variable (useful for preprocessing datasets).

Usage

```
delevels(x, levels, label = NULL)
```

Arguments

x	factor with several levels
levels	vector with the levels that will be replaced
label	the new label used for all levels examples (if NULL then "_OTHER" is assumed).

Value

Returns a factor with less levels.

Author(s)

Paulo Cortez http://www3.dsi.uminho.pt/pcortez

References

See fit.

See Also

fit and imputation.

Examples

```
f=factor(c("A","A","B","B","C","D","E"))
print(table(f))
f1=delevels(f,"A","a")
print(table(f1))
f2=delevels(f,c("C","D","E"),"CDE")
print(table(f2))
f3=delevels(f,c("B","C","D","E"))
print(table(f3))
```

fit *Fit a supervised data mining model (classification or regression) model*

Description

Fit a supervised data mining model (classification or regression) model. Wrapper function that allows to fit distinct data mining (14 classification and 15 regression) methods under the same coherent function structure. Also, it tunes the hyperparameters of the models (e.g. kknn, mlpe and ksvm) and performs some feature selection methods.

Usage

```
fit(x, data = NULL, model = "default", task = "default",
    search = "heuristic", mpar = NULL, feature = "none",
    scale = "default", transform = "none",
    created = NULL, fdebug = FALSE, ...)
```

Arguments

x — a symbolic description (formula) of the model to be fit.
If data=NULL it is assumed that x contains a formula expression with known variables (see first example below).

data — an optional data frame (columns denote attributes, rows show examples) containing the training data, when using a formula.

model — Typically this should be a character object with the model type name (data mining method, as explained in valid character options).

Valid character options are the typical R base learning functions, namely one of:

- naive most common class (classification) or mean output value (regression)
- ctree – conditional inference tree (classification and regression, uses ctree from party package)
- rpart or dt – decision tree (classification and regression, uses rpart from rpart package)
- kknn or knn – k-nearest neighbor (classification and regression, uses kknn from kknn package)
- mlp – multilayer perceptron with one hidden layer (classification and regression, uses nnet from nnet package)
- mlpe – multilayer perceptron ensemble (classification and regression, uses nnet from nnet package)
- ksvm or svm – support vector machine (classification and regression, uses ksvm from kernlab package)

- randomForest or randomforest – random forest algorithm (classification and regression, uses randomForest from randomForest package)
- bagging – bagging (classification, uses bagging from adabag package)
- boosting – boosting (classification, uses boosting from adabag package)
- lda – linear discriminant analysis (classification, uses lda from MASS package)
- multinom or lr – logistic regression (classification, uses multinom from nnet package)
- naiveBayes or naivebayes – naive bayes (classification, uses naiveBayes from e1071 package)
- qda – quadratic discriminant analysis (classification, uses qda from MASS package)
- mr – multiple regression (regression, equivalent to lm but uses nnet from nnet package with zero hidden nodes and linear output function)
- mars – multivariate adaptive regression splines (regression, uses mars from mda package)
- cubist – M5 rule-based model (regression, uses cubist from Cubist package)
- pcr – principal component regression (regression, uses pcr from pls package)
- plsr – partial least squares regression (regression, uses plsr from pls package)
- cppls – canonical powered partial least squares (regression, uses cppls from pls package)
- rvm – relevance vector machine (regression, uses rvm from kernlab package)

model can also be a list with the fields (see example below):

- $fit – a fit function that accepts the arguments x, data and ..., the goal is to accept here any R classification or regression model, mainly for its use within the mining or Importance functions, or to use a hyperparameter search (via search).
- $predict – a predict function that accepts the arguments object, newdata, this function should behave as any rminer prediction, i.e., return: a factor when task=="class"; a matrix with *Probabilities x Instances* when task=="prob"; and a vector when task=="reg".
- $name – optional field with the name of the method.

Note: current rminer version emphasizes the use of native fitting functions from their respective packages, since these functions contain several specific hyperparameters that can now be searched or set using the search or ... arguments. For compatibility with previous rminer versions, older model options are kept.

task data mining task. Valid options are:

- prob (or p) – classification with output probabilities (i.e. the sum of all outputs equals 1).
- class (or c) – classification with discrete outputs (factor)

- reg (or r) – regression (numeric output)
- default tries to guess the best task (prob or reg) given the model and output variable type (if factor then prob else reg)

search

used to tune hyperparameter(s) of the model, such as: kknn – number of neighbors (k); mlp or mlpe – number of hidden nodes (*size*) or *decay*; ksvm – gaussian kernel parameter (*sigma*); randomForest – mtry parameter). Valid options for a simpler search use:

- heuristic – simple heuristic, one search parameter (e.g. *k*=3 for knn, *size*=inputs/2 for mlp or *size*=10 if classification and inputs/2>10, *sigma* is set using kpar="automatic" and kernel="rbfdot" of ksvm). Note: all "heuristic" options (e.g. "heuristic5") only work currently for kknn, mlp or mlpe, ksvm (with kernel="rbfdot") and randomForest.
- heuristic5 – heuristic with a 5 range grid-search (e.g. seq(1,9,2) for kknn, seq(0,8,2) for mlp or mlpe, 2^seq(-15,3,4) for ksvm, 1:5 for randomRorest)
- heuristic10 – heuristic with a 10 range grid-search (e.g. seq(1,10,1) for kknn, seq(0,9,1) for mlp or mlpe, 2^seq(-15,3,2) for ksvm, 1:10 for randomRorest)
- UD, UD1 or UD2 – uniform design 2-Level with 13 (UD or UD2) or 21 (UD1) searches (only works for ksvm and kernel="rbfdot").
- a-vector – numeric vector with all hyperparameter values that will be searched within an internal grid-search (the number of searches is length(search) when convex=0)

A more complex but advised use of search is to use a list with:

- $smethod – type of search method. Valid options are (more options will be developed in next versions):
  - none – no search is executed, one single fit is performed.
  - matrix – matrix search (tests only n searches, all search parameters are of size n).
  - grid – normal grid search (tests all combinations of search parameters).
  - 2L - nested 2-Level grid search. First level range is set by $search and then the 2nd level performs a fine tuning, with length($search) searches around (original range/2) best value in first level (2nd level is only performed on numeric searches).
  - UD, UD1 or UD2 – uniform design 2-Level with 13 (UD or UD2) or 21 (UD1) searches (note: only works for model="ksvm" and kernel="rbfdot"). Under this option, $search should contain the first level ranges, such as c(-15,3,-5,15) for classification (*gamma* min and max, *C* min and max, after which a 2^ transform is applied) or c(-8,0,-1,6,-8,-1) for regression (last two values are *epsilon* min and max, after which a 2^ transform is applied).
- $search – a-list with all hyperparameter values to be searched or character with previous described options (e.g. "heuristic", "heuristic5", "UD"). If a character, then $smethod equal to "none" or "grid" or "UD" is automatically assumed.

- `$convex` – number that defines how many searches are performed after a local minimum/maximum is found (if >0, the search can be stopped without testing all grid-search values)
- `$method` – type of internal estimation method used during the search (see `method` argument of `mining` for details)
- `$metric` – used to compute a metric value during internal estimation. Can be a single character such as `"SAD"` or a list with all the arguments used by the `mmetric` function except `y` and `x`, such as `search$metric=list(metric="AUC",TC=3,D=0.7)`. See `mmetric` for more details.

Note: if `mpar` argument is used, then the `mpar` values are automatically fed into search. However, a direct use of the `search` argument is advised instead of `mpar`, since `search` is more flexible and powerful.

`mpar` Important note: this argument only is kept in this version due to compatibility with previous rminer versions. Instead of `mpar`, you should use the more flexible and powerful `search` argument.

vector with extra default (fixed) model parameters (used for modeling, search and feature selection) with:

- c(*vmethod*,*vpar*,`metric`) – generic use of mpar (including most models);
- c(*C*,*epsilon*,*vmethod*,*vpar*,`metric`) – if `ksvm` and C and epsilon are explicitly set;
- c(*nr*,*maxit*,*vmethod*,*vpar*,`metric`) – if `mlp` or `mlpe` and nr and maxit are explicitly set;

C and *epsilon* are default values for `svm` (if any of these is `=NA` then heuristics are used to set the value).
nr is the number of `mlp` runs or `mlpe` individual models, while *maxit* is the maximum number of epochs (if any of these is `=NA` then heuristics are used to set the value).
For help on *vmethod* and *vpar* see `mining`.
`metric` is the internal error function (e.g. used by search to select the best model), valid options are explained in `mmetric`. When `mpar=NULL` then default values are used. If there are `NA` values (e.g. `mpar=c(NA,NA)`) then default values are used.

`feature` feature selection and sensitivity analysis control. Valid `fit` function options are:

- `none` – no feature selection;
- a *fmethod* character value, such as `sabs` (see below);
- a-`vector` – vector with c(*fmethod*,*deletions*,*Runs*,*vmethod*,*vpar*,*defaultsearch*)
- a-`vector` – vector with c(*fmethod*,*deletions*,*Runs*,*vmethod*,*vpar*)

fmethod sets the type. Valid options are:

- `sbs` – standard backward selection;
- `sabs` – sensitivity analysis backward selection (faster);
- `sabsv` – equal to `sabs` but uses variance for sensitivity importance measure;
- `sabsr` – equal to `sabs` but uses range for sensitivity importance measure;

- sabsg – equal to sabs (uses gradient for sensitivity importance measure);

deletions is the maximum number of feature deletions (if -1 not used).
Runs is the number of runs for each feature set evaluation (e.g. 1).
For help on *vmethod* and *vpar* see mining.
defaultsearch is one hyperparameter used during the feature selection search, after selecting the best feature set then search is used (faster). If not defined, then search is used during feature selection (may be slow).
When feature is a vector then default values are used to fill missing values or NA values. Note: feature selection capabilities are expected to be enhanced in next rminer versions.

scale — if data needs to be scaled (i.e. for mlp or mlpe). Valid options are:

- default – uses scaling when needed (i.e. for mlp or mlpe)
- none – no scaling;
- inputs – standardizes (0 mean, 1 st. deviation) input attributes;
- all – standardizes (0 mean, 1 st. deviation) input and output attributes;

If needed, the predict function of rminer performs the inverse scaling.

transform — if the output data needs to be transformed (e.g. log transform). Valid options are:

- none – no transform;
- log – y=(log(y+1)) (the inverse function is applied in the predict function);
- positive – all predictions are positive (negative values are turned into zero);
- logpositive – both log and logpositive;

created — time stamp for the model. By default, the system time is used. Else, you can specify another time.

fdebug — if TRUE show some search details.

... — additional and specific parameters send to each fit function model (e.g. dt, randomforest, kernlab). A few examples:
– the rpart function is used for decision trees, thus you can have:
control=rpart.control(cp=.05) (see crossvaldata example).
– the ksvm function is used for support vector machines, thus you can change the kernel type: kernel="polydot" (see examples below).
Important note: if you use package functions and get an error, then try to explicitly define the package. For instance, you might need to use fit(*several-arguments*,control=Cubist::cubistControl()) instead of
fit(*several-arguments*,control=cubistControl()).

Details

Fits a classification or regression model given a data.frame (see [Cortez, 2010] for more details). The ... optional arguments should be used to fix values used by specific model functions (see examples). Notes:
- if there is an error in the fit, then a warning is issued (see example).
- the new search argument is very flexible and allows a powerful design of supervised learning

models.
- the search correct use is very dependent on the R learning base functions. For example, if you are tuning model="rpart" then read carefully the help of function rpart.
- mpar argument is only kept due to compatibility issues and should be avoided; instead, use the more flexible search.

Details about some models:

- Neural Network: mlp trains *nr* multilayer perceptrons (with *maxit* epochs, *size* hidden nodes and *decay* value according to the nnet function) and selects the best network according to minimum penalized error ($value). mlpe uses an ensemble of *nr* networks and the final prediction is given by the average of all outputs. To tune mlp or mlpe you can use the search parameter, which performs a grid search for *size* or *decay*.
- Support Vector Machine: svm adopts by default the gaussian (rbfdot) kernel. For classification tasks, you can use search to tune *sigma* (gaussian kernel parameter) and *C* (complexity parameter). For regression, the epsilon insensitive function is adopted and there is an additional hyperparameter *epsilon*.
- Other methods: Random Forest – if needed, you can tune several parameters, including the default mtry parameter adopted by search heuristics; k-nearest neighbor – search by default tunes *k*. The remaining models can also be tunned but a full definition of search is required (e.g. with $smethod, $search and other fields).

Value

Returns a model object. You can check all model elements with str(M), where M is a model object. The slots are:

- @formula – the x;
- @model – the model;
- @task – the task;
- @mpar – data.frame with the best model parameters (interpretation depends on model);
- @attributes – the attributes used by the model;
- @scale – the scale;
- @transform – the transform;
- @created – the date when the model was created;
- @time – computation effort to fit the model;
- @object – the R object model (e.g. rpart, nnet, ...);
- @outindex – the output index (of @attributes);
- @levels – if task=="prob"||task=="class" stores the output levels;

Note

See also http://www3.dsi.uminho.pt/pcortez/rminer.html

Author(s)

Paulo Cortez http://www3.dsi.uminho.pt/pcortez

References

- To check for more details about rminer and for citation purposes:
  P. Cortez.
  Data Mining with Neural Networks and Support Vector Machines Using the R/rminer Tool.
  In P. Perner (Ed.), Advances in Data Mining - Applications and Theoretical Aspects 10th Industrial Conference on Data Mining (ICDM 2010), Lecture Notes in Artificial Intelligence 6171, pp. 572-583, Berlin, Germany, July, 2010. Springer. ISBN: 978-3-642-14399-1.
  @Springer: `http://www.springerlink.com/content/e7u36014r04h0334`
  `http://www3.dsi.uminho.pt/pcortez/2010-rminer.pdf`

- For the grid search and other optimization methods:
  P. Cortez.
  Modern Optimization with R.
  Use R! series, Springer, September 2014, ISBN 978-3-319-08262-2.
  `http://www.springer.com/mathematics/book/978-3-319-08262-2`

- For the sabs feature selection:
  P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis.
  Modeling wine preferences by data mining from physicochemical properties.
  In Decision Support Systems, Elsevier, 47(4):547-553, 2009.
  `http://dx.doi.org/10.1016/j.dss.2009.05.016`

- For the uniform design details:
  C.M. Huang, Y.J. Lee, D.K.J. Lin and S.Y. Huang.
  Model selection for support vector machines via uniform design,
  In Computational Statistics & Data Analysis, 52(1):335-346, 2007.

See Also

`mining`, `predict.fit`, `mgraph`, `mmetric`, `savemining`, `CasesSeries`, `lforecast`, `holdout` and `Importance`. Check all rminer functions using: `help(package=rminer)`.

Examples

```
### dontrun is used when the execution of the example requires some computational effort.

### simple regression (with a formula) example.
x1=rnorm(200,100,20); x2=rnorm(200,100,20)
y=0.7*sin(x1/(25*pi))+0.3*sin(x2/(25*pi))
M=fit(y~x1+x2,model="mlpe")
new1=rnorm(100,100,20); new2=rnorm(100,100,20)
ynew=0.7*sin(new1/(25*pi))+0.3*sin(new2/(25*pi))
P=predict(M,data.frame(x1=new1,x2=new2,y=rep(NA,100)))
print(mmetric(ynew,P,"MAE"))

### simple classification example.
## Not run:
data(iris)
```

```
M=fit(Species~.,iris,model="rpart")
plot(M@object); text(M@object) # show model
P=predict(M,iris)
print(mmetric(iris$Species,P,"CONF"))
print(mmetric(iris$Species,P,"ALL"))
mgraph(iris$Species,P,graph="ROC",TC=2,main="versicolor ROC",
baseline=TRUE,leg="Versicolor",Grid=10)

M2=fit(Species~.,iris,model="ctree")
plot(M2@object) # show model
P2=predict(M2,iris)
print(mmetric(iris$Species,P2,"CONF"))

# ctree with different setup:
# (ctree_control is from the party package)
M3=fit(Species~.,iris,model="ctree",controls = party::ctree_control(testtype="MonteCarlo"))
plot(M3@object) # show model

## End(Not run)

### classification example with discrete classes, probabilities and holdout
## Not run:
data(iris)
H=holdout(iris$Species,ratio=2/3)
M=fit(Species~.,iris[H$tr,],model="ksvm",task="class")
M2=fit(Species~.,iris[H$tr,],model="ksvm",task="prob")
P=predict(M,iris[H$ts,])
P2=predict(M2,iris[H$ts,])
print(mmetric(iris$Species[H$ts],P,"CONF"))
print(mmetric(iris$Species[H$ts],P2,"CONF"))
print(mmetric(iris$Species[H$ts],P,"CONF",TC=1))
print(mmetric(iris$Species[H$ts],P2,"CONF",TC=1))
print(mmetric(iris$Species[H$ts],P2,"AUC"))

### exploration of some rminer classification models:
models=c("lda","naiveBayes","kknn","randomForest")
for(m in models)
 { cat("model:",m,"\n")
   M=fit(Species~.,iris[H$tr,],model=m)
   P=predict(M,iris[H$ts,])
   print(mmetric(iris$Species[H$ts],P,"AUC")[[1]])
 }

## End(Not run)

### classification example with hyperparameter selection
### SVM
## Not run:
data(iris)
# large list of SVM configurations:
# SVM with kpar="automatic" sigma rbfdot kernel estimation and default C=1:
M=fit(Species~.,iris,model="ksvm")
print(M@mpar) # model hyperparameters/arguments
```

```
# SVM with C=3, sigma=2^-7
M=fit(Species~.,iris,model="ksvm",C=3,kpar=list(sigma=2^-7))
print(M@mpar)
# SVM with different kernels:
M=fit(Species~.,iris,model="ksvm",kernel="polydot",kpar="automatic")
print(M@mpar)
# fit already has a scale argument, thus the only way to fix scale of "tanhdot"
# is to use the special search argument with the "none" method:
search=list(smethod="none",search=list(scale=2,offset=2))
M=fit(Species~.,iris,model="ksvm",kernel="tanhdot",search=search)
print(M@mpar)
# 10 grid search values for sigma, rbfdot kernel (fdebug is used only for more verbose):
M=fit(Species~.,iris,model="ksvm",search="heuristic10",fdebug=TRUE)
print(M@mpar)
# identical search under a different argument use:
M=fit(Species~.,iris,model="ksvm",search=2^seq(-15,3,2),fdebug=TRUE)
print(M@mpar)
# UD for sigma and C, rbfdot kernel, two level of grid searches, exponential (2^x) search scale:
M=fit(Species~.,iris,model="ksvm",search="UD",fdebug=TRUE)
print(M@mpar)
M=fit(Species~.,iris,model="ksvm",search="UD1",fdebug=TRUE)
print(M@mpar)
M=fit(Species~.,iris,model="ksvm",search=2^seq(-15,3,2),fdebug=TRUE)
print(M@mpar)
# now the more powerful search argument is used for modeling SVM:
# grid 3 x 3 search:
search=list(smethod="grid",search=list(sigma=2^c(-15,-5,3),C=2^c(-5,0,15)),convex=0,
            metric="AUC",method=c("kfold",3,12345))
print(search)
M=fit(Species~.,iris,model="ksvm",search=search,fdebug=TRUE)
print(M@mpar)
# identical search with different argument smethod="matrix"
search$smethod="matrix"
search$search=list(sigma=rep(2^c(-15,-5,3),times=3),C=rep(2^c(-5,0,15),each=3))
print(search)
M=fit(Species~.,iris,model="ksvm",search=search,fdebug=TRUE)
print(M@mpar)
# search for best kernel (only works for kpar="automatic"):
search=list(smethod="grid",search=list(kernel=c("rbfdot","laplacedot","polydot","vanilladot")),
            convex=0,metric="AUC",method=c("kfold",3,12345))
print(search)
M=fit(Species~.,iris,model="ksvm",search=search,fdebug=TRUE)
print(M@mpar)
# search for best parameters of "rbfdot" or "laplacedot" (which use same kpar):
search$search=list(kernel=c("rbfdot","laplacedot"),sigma=2^seq(-15,3,5))
print(search)
M=fit(Species~.,iris,model="ksvm",search=search,fdebug=TRUE)
print(M@mpar)

### randomForest
# search for mtry and ntree
search=list(smethod="grid",search=list(mtry=c(1,2,3),ntree=c(100,200,500)),
            convex=0,metric="AUC",method=c("kfold",3,12345))
```

```
print(search)
M=fit(Species~.,iris,model="randomForest",search=search,fdebug=TRUE)
print(M@mpar)

### some MLP fitting examples:
# simplest use:
M=fit(Species~.,iris,model="mlpe")
print(M@mpar) # hidden nodes and number of ensemble mlps
# setting some nnet parameters:
M=fit(Species~.,iris,model="mlpe",size=3,decay=0.1,maxit=100,rang=0.9)
print(M@mpar) # mlpe hyperparameters
# MLP, 5 grid search fdebug is only used to put some verbose in the console:
M=fit(Species~.,iris,model="mlpe",search="heuristic5",fdebug=TRUE)
print(M@mpar)
# identical search, using explicit search:
M=fit(Species~.,iris,model="mlpe",search=seq(0,8,2),fdebug=TRUE)
print(M@mpar)
# previous searches used a random holdout (seed=NULL), now a fixed seed (123) is used:
search=list(smethod="grid",search=list(size=seq(0,8,2)),convex=0,metric="AUC",
            method=c("holdout",2/3,123))
print(search)
M=fit(Species~.,iris,model="mlpe",search=search,fdebug=TRUE)
print(M@mpar)
# faster and greedy grid search:
search$convex=1;search$search=list(size=0:9)
print(search)
M=fit(Species~.,iris,model="mlpe",search=search,fdebug=TRUE)
print(M@mpar)
# 2 level grid with total of 5 searches
#  note of caution: some "2L" ranges may lead to non integer (e.g. 1.3) values at
#  the 2nd level search. And some R functions crash if non integer values are used for
#  integer parameters.
search$smethod="2L";search$convex=0;search$search=list(size=c(4,8,12))
print(search)
M=fit(Species~.,iris,model="mlpe",search=search,fdebug=TRUE)
print(M@mpar)

## End(Not run)

### example of an error (warning) generated using fit:
## Not run:
data(iris)
# size needs to be a positive integer, thus 0.1 leads to an error:
M=fit(Species~.,iris,model="mlp",size=0.1)
print(M@object)

## End(Not run)

### exploration of some rminer regression models:
## Not run:
data(sa_ssin)
H=holdout(sa_ssin$y,ratio=2/3,seed=12345)
models=c("mr","ctree","mars","cubist","rvm")
```

```
for(m in models)
 { cat("model:",m,"\n")
   M=fit(y~.,sa_ssin[H$tr,],model=m)
   P=predict(M,sa_ssin[H$ts,])
   print(mmetric(sa_ssin$y[H$ts],P,"MAE"))
 }

## End(Not run)

### regression example with hyperparameter detection:
## Not run:
data(sa_ssin)
# some SVM experiments:
# default SVM:
M=fit(y~.,data=sa_ssin,model="svm")
print(M@mpar)
# SVM with (Cherkassy and Ma, 2004) heuristics to set C and epsilon:
M=fit(y~.,data=sa_ssin,model="svm",C=NA,epsilon=NA)
print(M@mpar)
# SVM with Uniform Design set sigma, C and epsilon:
M=fit(y~.,data=sa_ssin,model="ksvm",search="UD",fdebug=TRUE)
print(M@mpar)

# sensitivity analysis feature selection
M=fit(y~.,data=sa_ssin,model="ksvm",search="heuristic5",feature="sabs")
print(M@mpar)
print(M@attributes) # selected attributes (1, 2 and 3 are the relevant inputs)

# example that shows how transform works:
M=fit(y~.,data=sa_ssin,model="mr") # linear regression
P=predict(M,data.frame(x1=-1000,x2=0,x3=0,x4=0,y=NA)) # P should be negative
print(P)
M=fit(y~.,data=sa_ssin,model="mr",transform="positive")
P=predict(M,data.frame(x1=-1000,x2=0,x3=0,x4=0,y=NA)) # P is not negative
print(P)

## End(Not run)

### pure classification example with a generic R model ###
## Not run:
### nnet is adopted here but virtually ANY fitting function/package could be used:

# since the default nnet prediction is to provide probabilities, there is
# a need to create this "wrapping" function:
predictprob=function(object,newdata)
{ predict(object,newdata,type="class") }
# list with a fit and predict function:
# nnet::nnet (package::function)
model=list(fit=nnet::nnet,predict=predictprob,name="nnet")
data(iris)
# note that size is not a fit parameter and it is sent directly to nnet:
M=fit(Species~.,iris,model=model,size=3,task="class")
P=predict(M,iris)
```

```
print(P)

## End(Not run)
```

`holdout` *Computes indexes for holdout data split into training and test sets.*

Description

Computes indexes for holdout data split into training and test sets.

Usage

```
holdout(y, ratio = 2/3, internalsplit = FALSE, mode = "stratified", iter = 1,
               seed = NULL, window=10, increment=1)
```

Arguments

`y` desired target: numeric vector; or factor – then a stratified holdout is applied (i.e. the proportions of the classes are the same for each set).

`ratio` split ratio (in percentage – sets the training set size; or in total number of examples – sets the test set size).

`internalsplit` if `TRUE` then the training data is further split into training and validation sets. The same `ratio` parameter is used for the internal split.

`mode` sampling mode. Options are:

- `stratified` – stratified randomized holdout if `y` is a factor; else it behaves as standard randomized holdout;
- `random` – standard randomized holdout;
- `order` – static mode, where the first examples are used for training and the later ones for testing (useful for time series data);
- `rolling` – rolling window, also known as sliding window (e.g. useful for stock market prediction), similar to `order` except that `window` is the window size, `iter` is the rolling iteration and `increment` is the number of samples slided at each iteration. In each iteration, the training set size is fixed to `window`, while the test set size is equal to `ratio` except for the last iteration (where it may be smaller).
- `incremental` – incremental retraining mode, also known as growing windows, similar to `order` except that `window` is the initial window size, `iter` is the incremental iteration and `increment` is the number of samples added at each iteration. In each iteration, the training set size grows (+increment), while the test set size is equal to `ratio` except for the last iteration (where it may be smaller).

`iter` iteration of the incremental retraining mode (only used when `mode="rolling"` or `"incremental"`, typically `iter` is set within a cycle, see the example below).

seed	if NULL then a random seed is used; else a fixed seed is adopted (will return always the same result for the same seed).
window	training window size (if mode="rolling") or initial training window size (if mode="incremental").
increment	number of samples added to the training window at each iteration (if mode="incremental" or mode="rolling").

Details

Computes indexes for holdout data split into training and test sets.

Value

A list with the components:

- $tr – numeric vector with the training examples indexes;
- $ts – numeric vector with the test examples indexes;
- $itr – numeric vector with the internal training examples indexes;
- $val – numeric vector with the internal validation examples indexes;

Author(s)

Paulo Cortez http://www3.dsi.uminho.pt/pcortez

References

See fit.

See Also

fit, predict.fit, mining, mgraph, mmetric, savemining, Importance.

Examples

```
### simple examples:
# preserves order, last two elements go into test set
H=holdout(1:10,ratio=2,internal=TRUE,mode="order")
print(H)
# no seed or NULL returns different splits:
H=holdout(1:10,ratio=2/3,mode="random")
print(H)
H=holdout(1:10,ratio=2/3,mode="random",seed=NULL)
print(H)
# same seed returns identical split:
H=holdout(1:10,ratio=2/3,mode="random",seed=12345)
print(H)
H=holdout(1:10,ratio=2/3,mode="random",seed=12345)
print(H)

### classification example
```

```
data(iris)
# random stratified holdout
H=holdout(iris$Species,ratio=2/3,mode="stratified")
print(table(iris[H$tr,]$Species))
print(table(iris[H$ts,]$Species))
M=fit(Species~.,iris[H$tr,],model="rpart") # training data only
P=predict(M,iris[H$ts,]) # test data
print(mmetric(iris$Species[H$ts],P,"CONF"))

### regression example with incremental and rolling window holdout:
## Not run:
ts=c(1,4,7,2,5,8,3,6,9,4,7,10,5,8,11,6,9)
d=CasesSeries(ts,c(1,2,3))
print(d) # with 14 examples
# incremental holdout example (growing window)
for(b in 1:4) # iterations
  {
   H=holdout(d$y,ratio=4,mode="incremental",iter=b,window=5,increment=2)
   M=fit(y~.,d[H$tr,],model="mlpe",search=2)
   P=predict(M,d[H$ts,])
   cat("batch :",b,"TR from:",H$tr[1],"to:",H$tr[length(H$tr)],"size:",length(H$tr),
       "TS from:",H$ts[1],"to:",H$ts[length(H$ts)],"size:",length(H$ts),
       "mae:",mmetric(d$y[H$ts],P,"MAE"),"\n")
  }
# rolling holdout example (sliding window)
for(b in 1:4) # iterations
  {
   H=holdout(d$y,ratio=4,mode="rolling",iter=b,window=5,increment=2)
   M=fit(y~.,d[H$tr,],model="mlpe",search=2)
   P=predict(M,d[H$ts,])
   cat("batch :",b,"TR from:",H$tr[1],"to:",H$tr[length(H$tr)],"size:",length(H$tr),
       "TS from:",H$ts[1],"to:",H$ts[length(H$ts)],"size:",length(H$ts),
       "mae:",mmetric(d$y[H$ts],P,"MAE"),"\n")
  }

## End(Not run)
```

`Importance` *Measure input importance (including sensitivity analysis) given a supervised data mining model.*

Description

Measure input importance (including sensitivity analysis) given a supervised data mining model.

Usage

```
Importance(M, data, RealL = 7, method = "1D-SA", measure = "AAD",
           sampling = "regular", baseline = "mean", responses = TRUE,
           outindex = NULL, task = "default", PRED = NULL,
           interactions = NULL, Aggregation = -1, LRandom = -1,
           MRandom = "discrete", Lfactor = FALSE)
```

Arguments

`M` fitted model, typically is the object returned by `fit`. Can also be any fitted model (i.e. not from rminer), provided that the predict function PRED is defined (see examples for details).

`data` training data (the same data.frame that was used to fit the model, currently only used to add data histogram to VEC curve).

`RealL` the number of sensitivity analysis levels (e.g. 7). Note: you need to use `RealL`>=2.

`method` input importance method. Options are:

- 1D-SA – 1 dimensional sensitivity analysis, very fast, sets interactions to NULL.
- sens or SA – sensitivity analysis. There are some extra variants: sensa – equal to `sens` but also sets `measure="AAD"`; sensv – sets `measure="variance"`; sensg – sets `measure="gradient"`; sensr – sets `measure="range"`. if interactions is not null, then GSA is assumed, else 1D-SA is assumed.
- DSA – Data-based SA (good option if input interactions need to be detected).
- MSA – Monte-Carlo SA.
- CSA – Cluster-based SA.
- GSA – Global SA (very slow method, particularly if the number of inputs is large, should be avoided).
- randomForest – uses method of Leo Breiman (type=1), only makes sense when M is a randomRorest.

`measure` sensitivity analysis measure (used to measure input importance). Options are:

- AAD – average absolute deviation from the median.
- gradient – average absolute gradient (y_i+1-y_i) of the responses.
- variance – variance of the responses.
- range – maximum - minimum of the responses.

`sampling` for numeric inputs, the sampling scan function. Options are:

- regular – regular sequence (uniform distribution), do not change this value, kept here only due to compatibility issues.

`baseline` baseline vector used during the sensitivity analysis. Options are:

- mean – uses a vector with the mean values of each attribute from `data`.
- median – uses a vector with the median values of each attribute from `data`.
- a data.frame with the baseline example (should have the same attribute names as `data`).

`responses` if `TRUE` then all sensitivity analysis responses are stored and returned.

`outindex` the output index (column) of `data` if `M` is not a model object (returned by fit).

`task` the `task` as defined in `fit` if `M` is not a model object (returned by fit).

`PRED` the prediction function of `M`, if `M` is not a model object (returned by fit). Note: this function should behave like the rminer `predict-methods`, i.e. return a numeric vector in case of regression; a matrix of examples (rows) vs probabilities (columns) (`task="prob"`) or a factor (`task="class"`) in case of classification.

`interactions` numeric vector with the attributes (columns) used by Ith-D sensitivity analysis (2-D or higher, "GSA" method):

- if `NULL` then only a 1-D sensitivity analysis is performed.
- if `length(interactions)==1` then a "special" 2-D sensitivity analysis is performed using the index of interactions versus all remaining inputs. Note: the $sresponses[[interactions]] will be empty (in `vecplot` do not use `xval =interactions`).
- if `length(interactions)>1` then a full Ith-D sensitivity analysis is performed, where I=length(interactions). Note: Computational effort can highly increase if I is too large, i.e. O(RealL^I). Also, you need to preprocess the returned list (e.g. using `avg_imp`) to use the `vecplot` function (see the examples).

`Aggregation` numeric value that sets the number of multi-metric aggregation function (used only for "DSA", ""). Options are:

- -1 – the default value that should work in most cases (if regression, sets Aggregation=3, else if classification then sets Aggregation=1).
- 1 – value that should work for classification (only use the average of all sensitivity values).
- 3 – value that should work for regression (use 3 metrics, the minimum, average and maximum of all sensitivity values).

`LRandom` number of samples used by DSA and MSA methods. The default value is -1, which means: use a number equal to training set size. If a different value is used (1<= value <= number of training samples), then LRandom samples are randomly selected.

`MRandom` sampling type used by MSA: "discrete" (default discrete uniform distribution) or "continuous" (from continuous uniform distribution).

`Lfactor` sets the maximum number of sensitivity levels for discrete inputs. if FALSE then a maximum of up to RealL levels are used (most frequent ones), else (TRUE) then all levels of the input are used in the SA analysis.

Details

This function provides several algorithms for measuring input importance of supervised data mining models and the average effect of a given input (or pair of inputs) in the model. A particular emphasis is given on sensitivity analysis (SA), which is a simple method that measures the effects on the output of a given model when the inputs are varied through their range of values. Check the references for more details.

Value

A `list` with the components:

- $value – numeric vector with the computed sensitivity analysis measure for each attribute.

- $imp – numeric vector with the relative importance for each attribute (only makes sense for 1-D analysis).
- $sresponses – vector list as described in the Value documentation of mining.
- $data – if DSA or MSA, store the used data samples, needed for visualizations made by vecplot.
- $method – SA method
- $measure – SA measure
- $agg – Aggregation value
- $nclasses – if task="prob" or "class", the number of output classes, else nclasses=1
- $inputs – indexes of the input attributes
- $Llevels – sensitivity levels used for each attribute (NA means output attribute)
- $interactions – which attributes were interacted when method=GSA.

Note

See also http://www3.dsi.uminho.pt/pcortez/rminer.html

Author(s)

Paulo Cortez http://www3.dsi.uminho.pt/pcortez

References

- To cite the Importance function, sensitivity analysis methods or synthetic datasets, please use:
  P. Cortez and M.J. Embrechts.
  Using Sensitivity Analysis and Visualization Techniques to Open Black Box Data Mining Models.
  In Information Sciences, Elsevier, 225:1-17, March 2013.
  http://dx.doi.org/10.1016/j.ins.2012.10.039

See Also

vecplot, fit, mining, mgraph, mmetric, savemining.

Examples

```
### dontrun is used when the execution of the example requires some computational effort.

### 1st example, regression, 1-D sensitivity analysis
## Not run:
data(sa_ssin) # x1 should account for 55
M=fit(y~.,sa_ssin,model="ksvm")
I=Importance(M,sa_ssin,method="1D-SA") # 1-D SA, AAD
print(round(I$imp,digits=2))

L=list(runs=1,sen=t(I$imp),sresponses=I$sresponses)
```

```
mgraph(L,graph="IMP”,leg=names(sa_ssin),col="gray”,Grid=10)
mgraph(L,graph="VEC”,xval=1,Grid=10,data=sa_ssin,
   main="VEC curve for x1 influence on y”) # or:
vecplot(I,xval=1,Grid=10,data=sa_ssin,datacol="gray”,
   main="VEC curve for x1 influence on y”) # same graph
vecplot(I,xval=c(1,2,3),pch=c(1,2,3),Grid=10,
leg=list(pos="bottomright”,leg=c(”x1”,”x2”,”x3”))) # all x1, x2 and x3 VEC curves

## End(Not run)

### 2nd example, regression, DSA sensitivity analysis:
## Not run:
I2=Importance(M,sa_ssin,method="DSA”)
print(I2)
# influence of x1 and x2 over y
vecplot(I2,graph="VEC”,xval=1) # VEC curve
vecplot(I2,graph="VECB”,xval=1) # VEC curve with boxplots
vecplot(I2,graph="VEC3”,xval=c(1,2)) # VEC surface
vecplot(I2,graph="VECC”,xval=c(1,2)) # VEC contour

## End(Not run)

### 3th example, classification (pure class labels, task="cla”), DSA:
## Not run:
data(sa_int2_3c) # pair (x1,x2) is more relevant than x3, all x1,x2,x3 affect y,
                 # x4 has a null effect.
M2=fit(y~.,sa_int2_3c,model="mlpe”,task="class”)
I4=Importance(M2,sa_int2_3c,method="DSA”)
# VEC curve (should present a kind of ”saw” shape curve) for class B (TC=2):
vecplot(I4,graph="VEC”,xval=2,cex=1.2,TC=2,
 main="VEC curve for x2 influence on y (class B)”,xlab="x2”)
# same VEC curve but with boxplots:
vecplot(I4,graph="VECB”,xval=2,cex=1.2,TC=2,
 main="VEC curve with box plots for x2 influence on y (class B)”,xlab="x2”)

## End(Not run)

### 4th example, regression, DSA:
## Not run:
data(sa_psin)
# same model from Table 1 of the reference:
M3=fit(y~.,sa_psin,model="ksvm”,search=2^-2,C=2^6.87,epsilon=2^-8)
# in this case: Aggregation is the same as NY
I5=Importance(M3,sa_psin,method="DSA”,Aggregation=3)
# 2D analysis (check reference for more details), RealL=L=7:
# need to aggregate results into a matrix of SA measure
cm=agg_matrix_imp(I5)
print(”show Table 8 DSA results (from the reference):”)
print(round(cm$m1,digits=2))
print(round(cm$m2,digits=2))
# show most relevant (darker) input pairs, in this case (x1,x2) > (x1,x3) > (x2,x3)
# to build a nice plot, a fixed threshold=c(0.05,0.05) is used. note that
# in the paper and for real data, we use threshold=0.1,
# which means threshold=rep(max(cm$m1,cm$m2)*threshold,2)
fcm=cmatrixplot(cm,threshold=c(0.05,0.05))
# 2D analysis using pair AT=c(x1,x2') (check reference for more details), RealL=7:
# nice 3D VEC surface plot:
vecplot(I5,xval=c(1,2),graph="VEC3",xlab="x1",ylab="x2",zoom=1.1,
 main="VEC surface of (x1,x2') influence on y")
# same influence but know shown using VEC contour:
par(mar=c(4.0,4.0,1.0,0.3)) # change the graph window space size
vecplot(I5,xval=c(1,2),graph="VECC",xlab="x1",ylab="x2",
 main="VEC surface of (x1,x2') influence on y")
# slower GSA:
I6=Importance(M3,sa_psin,method="GSA",interactions=1:4)
cm2=agg_matrix_imp(I6)
# compare cm2 with cm1, almost identical:
print(round(cm2$m1,digits=2))
print(round(cm2$m2,digits=2))
fcm2=cmatrixplot(cm2,threshold=0.1)

## End(Not run)

### If you want to use Importance over your own model (different than rminer ones):
# 1st example, regression, uses the theoretical sin1reg function: x1=70% and x2=30%
data(sin1reg)
mypred=function(M,data)
{ return (M[1]*sin(pi*data[,1]/M[3])+M[2]*sin(pi*data[,2]/M[3])) }
M=c(0.7,0.3,2000)
# 4 is the column index of y
I=Importance(M,sin1reg,method="sens",measure="AAD",PRED=mypred,outindex=4)
print(I$imp) # x1=72.3% and x2=27.7%
L=list(runs=1,sen=t(I$imp),sresponses=I$sresponses)
mgraph(L,graph="IMP",leg=names(sin1reg),col="gray",Grid=10)
mgraph(L,graph="VEC",xval=1,Grid=10) # equal to:
par(mar=c(2.0,2.0,1.0,0.3)) # change the graph window space size
vecplot(I,graph="VEC",xval=1,Grid=10,main="VEC curve for x1 influence on y:")

### 2nd example, 3-class classification for iris and lda model:
## Not run:
data(iris)
library(MASS)
predlda=function(M,data) # the PRED function
{ return (predict(M,data)$posterior) }
LDA=lda(Species ~ .,iris, prior = c(1,1,1)/3)
# 4 is the column index of Species
I=Importance(LDA,iris,method="1D-SA",PRED=predlda,outindex=4)
vecplot(I,graph="VEC",xval=1,Grid=10,TC=1,
main="1-D VEC for Sepal.Lenght (x-axis) influence in setosa (prob.)")

## End(Not run)

### 3rd example, binary classification for setosa iris and lda model:
## Not run:
iris2=iris;iris2$Species=factor(iris$Species=="setosa")
predlda2=function(M,data) # the PRED function
```

```
{ return (predict(M,data)$class) }
LDA2=lda(Species ~ .,iris2)
I=Importance(LDA2,iris2,method="1D-SA",PRED=predlda2,outindex=4)
vecplot(I,graph="VEC",xval=1,
main="1-D VEC for Sepal.Lenght (x-axis) influence in setosa (class)",Grid=10)

## End(Not run)
```

`imputation` *Missing data imputation (e.g. substitution by value or hotdeck method).*

Description

Missing data imputation (e.g. substitution by value or hotdeck method).

Usage

```
imputation(imethod = "value", D, Attribute = NULL, Missing = NA, Value = 1)
```

Arguments

`imethod` imputation method type:

- `value` – substitutes missing data by `Value` (with single element or several elements);
- `hotdeck` – searches first the most similar example (i.e. using a k-nearest neighbor method – `knn`) in the dataset and replaces the missing data by the value found in such example;

`D` dataset with missing data (data.frame)

`Attribute` if `NULL` then all attributes (data columns) with missing data are replaced. Else, `Attribute` is the attribute number (numeric) or name (character).

`Missing` missing data symbol

`Value` the substitution value (if `imethod=value`) or number of neighbors (*k* of `knn`).

Details

Check the references.

Value

A data.frame without missing data.

Author(s)

Paulo Cortez `http://www3.dsi.uminho.pt/pcortez`

References

M. Brown and J. Kros.
Data mining and the impact of missing data.
In Industrial Management & Data Systems, 103(8):611-621, 2003.

See Also

`fit` and `delevels`.

Examples

```
d=matrix(ncol=5,nrow=5)
d[1,]=c(5,4,3,2,1)
d[2,]=c(4,3,4,3,4)
d[3,]=c(1,1,1,1,1)
d[4,]=c(4,NA,3,4,4)
d[5,]=c(5,NA,NA,2,1)
d=data.frame(d); d[,3]=factor(d[,3])
print(d)
print(imputation(”value”,d,3,Value=”3”))
print(imputation(”value”,d,2,Value=median(na.omit(d[,2]))))
print(imputation(”value”,d,2,Value=c(1,2)))
print(imputation(”hotdeck”,d,”X2”,Value=1))
print(imputation(”hotdeck”,d,Value=1))

## Not run:
# hotdeck 1-nearest neighbor substitution on a real dataset:
d=read.table(
  file=”http://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data”,
   sep=”,”,na.strings=”?”)
print(summary(d))
d2=imputation(”hotdeck”,d,Value=1)
print(summary(d2))
par(mfrow=c(2,1))
hist(d$V26)
hist(d2$V26)
par(mfrow=c(1,1)) # reset mfrow

## End(Not run)
```

lforecast *Compute long term forecasts.*

Description

Performs multi-step forecasts by iteratively using 1-ahead predictions as inputs

Usage

```
lforecast(M, data, start, horizon)
```

Arguments

M	fitted model, the object returned by fit.
data	training data, typically built using CasesSeries.
start	starting period (when out-of-samples start).
horizon	number of multi-step predictions.

Details

Check the reference for details.

Value

Returns a numeric vector with the multi-step predictions.

Author(s)

Paulo Cortez http://www3.dsi.uminho.pt/pcortez

References

- To check for more details:
  P. Cortez.
  Sensitivity Analysis for Time Lag Selection to Forecast Seasonal Time Series using Neural Networks and Support Vector Machines.
  In Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN 2010), pp. 3694-3701, Barcelona, Spain, July, 2010. IEEE Computer Society, ISBN: 978-1-4244-6917-8 (DVD edition).
  http://dx.doi.org/10.1109/IJCNN.2010.5596890

See Also

fit, CasesSeries, predict.fit, mgraph.

Examples

```
ts=c(1,4,7,2,5,8,3,6,9,4,7,10,5,8,11,6,9)
d=CasesSeries(ts,c(1,2,3))
M=fit(y~.,d[1:7,],model="mlpe",search=2)
P1=predict(M,d[8:14,]) # single-step predictions
P2=lforecast(M,d,8,7) # multi-step predictions, horizon=7
print(mmetric(d$y[8:14],P1,"MAE"))
print(mmetric(d$y[8:14],P2,"MAE"))
L=vector("list",2); pred=vector("list",1);test=vector("list",1)
pred[[1]]=P1; test[[1]]=d$y[8:14]; L[[1]]=list(pred=pred,test=test,runs=1)
```

```
pred[[1]]=P2; test[[1]]=d$y[8:14]; L[[2]]=list(pred=pred,test=test,runs=1)
mgraph(L,graph="REG",Grid=10,leg=c("y","P1","P2"),col=c("black","cyan","blue"))
mgraph(L,graph="RSC",Grid=10,leg=c("P1","P2"),col=c("cyan","blue"))
```

mgraph *Mining graph function*

Description

Plots a graph given a mining list, list of several mining lists or given the pair y - target and x - predictions.

Usage

```
mgraph(y, x = NULL, graph, leg = NULL, xval = -1, PDF = "", PTS = -1,
       size = c(5, 5), sort = TRUE, ranges = NULL, data = NULL,
       digits = NULL, TC = -1, intbar = TRUE, lty = 1, col = "black",
       main = "", metric = "MAE", baseline = FALSE, Grid = 0,
       axis = NULL, cex = 1)
```

Arguments

y — if there are predictions (!is.null(x)), y should be a numeric vector or factor with the target desired responses (or output values).
Else, y should be a list returned by the mining function or a vector list with several mining lists.

x — the predictions (should be a numeric vector if task="reg", matrix if task="prob" or factor if task="class" (use if y is not a list).

graph — type of graph. Options are:

- ROC – ROC curve (classification);
- LIFT – LIFT accumulative curve (classification);
- IMP – relative input importance barplot;
- REC – REC curve (regression);
- VEC – variable effect curve;
- RSC – regression scatter plot;
- REP – regression error plot;
- REG – regression plot;
- DLC – distance line comparison (for comparing errors in one line);

leg — legend of graph:

- if NULL – not used;
- if -1 and graph="ROC" or "LIFT" – the target class name is used;
- if -1 and graph="REG" – leg=c("Target","Predictions");
- if -1 and graph="RSC" – leg=c("Predictions");
- if vector with "character" type (text) – the text of the legend;

- if is list – `$leg` = vector with the text of the legend and `$pos` is the position of the legend (e.g. "top" or c(4,5));

`xval` auxiliary value, used by some graphs:

- `VEC` – if -1 means perform several 1-D sensitivity analysis VEC curves, one for each attribute, if >0 means the attribute index (e.g. 1).
- `ROC` or `LIFT` or `REC` – if -1 then `xval=1`. For these graphs, `xval` is the maximum x-axis value.
- `IMP` – `xval` is the x-axis value for the legend of the attributes.
- `REG` – `xval` is the set of plotted examples (e.g. 1:5), if -1 then all examples are used.
- `DLC` – `xval` is the `val` of the `mmetric` function.

`PDF` if `""` then the graph is plotted on the screen, else the graph is saved into a pdf file with the name set in this argument.

`PTS` number of points in each line plot. If -1 then `PTS=11` (for `ROC`, `REC` or `LIFT`) or `PTS=6` (`VEC`).

`size` size of the graph, c(width,height), in inches.

`sort` if TRUE then sorts the data (works only for some graphs, e.g. `VEC`, `IMP`, `REP`).

`ranges` matrix with the attribute minimum and maximum ranges (only used by `VEC`).

`data` the training data, for plotting histograms and getting the minimum and maximum attribute ranges if not defined in ranges (only used by `VEC`).

`digits` the number of digits for the axis, can also be defined as c(x-axis digits,y-axis digits) (only used by `VEC`).

`TC` target class (for multi-class classification class) within 1,...,*Nc*, where *Nc* is the number of classes. If multi-class and TC==-1 then TC is set to the index of the last class.

`intbar` if 95% confidence interval bars (according to t-student distribution) should be plotted as whiskers.

`lty` the same `lty` argument of the `par` function.

`col` color, as defined in the `par` function.

`main` the title of the graph, as defined in the `plot` function.

`metric` the error metric, as defined in `mmetric` (used by `DLC`).

`baseline` if the baseline should be plotted (used by `ROC` and `LIFT`).

`Grid` if >1 then there are GRID light gray squared grid lines in the plot.

`axis` Currently only used by `IMP`: numeric vector with the axis numbers (1 – bottom, 3 – top). If `NULL` then `axis=c(1,3)`.

`cex` label font size

Details

Plots a graph given a `mining` list, list of several mining lists or given the pair y - target and x - predictions.

Value

A graph (in screen or pdf file).

Note

See also http://www3.dsi.uminho.pt/pcortez/rminer.html

Author(s)

Paulo Cortez http://www3.dsi.uminho.pt/pcortez

References

- To check for more details about rminer and for citation purposes:
  P. Cortez.
  Data Mining with Neural Networks and Support Vector Machines Using the R/rminer Tool.
  In P. Perner (Ed.), Advances in Data Mining - Applications and Theoretical Aspects 10th Industrial Conference on Data Mining (ICDM 2010), Lecture Notes in Artificial Intelligence 6171, pp. 572-583, Berlin, Germany, July, 2010. Springer. ISBN: 978-3-642-14399-1.
  @Springer: http://www.springerlink.com/content/e7u36014r04h0334
  http://www3.dsi.uminho.pt/pcortez/2010-rminer.pdf

See Also

fit, predict.fit, mining, mmetric, savemining and Importance.

Examples

```
### regression
y=c(1,5,10,11,7,3,2,1);x=rnorm(length(y),0,1.0)+y
mgraph(y,x,graph="RSC",Grid=10,col=c("blue"))
mgraph(y,x,graph="REG",Grid=10,lty=1,col=c("black","blue"),
       leg=list(pos="topleft",leg=c("target","predictions")))
mgraph(y,x,graph="REP",Grid=10)
mgraph(y,x,graph="REP",Grid=10,sort=FALSE)
x2=rnorm(length(y),0,1.2)+y;x3=rnorm(length(y),0,1.4)+y;
L=vector("list",3); pred=vector("list",1); test=vector("list",1);
pred[[1]]=y; test[[1]]=x; L[[1]]=list(pred=pred,test=test,runs=1)
test[[1]]=x2; L[[2]]=list(pred=pred,test=test,runs=1)
test[[1]]=x3; L[[3]]=list(pred=pred,test=test,runs=1)
mgraph(L,graph="DLC",metric="MAE",leg=c("x1","x2","x3"),main="MAE errors")

### regression example with mining
## Not run:
data(sin1reg)
M1=mining(y~.,sin1reg[,c(1,2,4)],model="mr",Runs=5)
M2=mining(y~.,sin1reg[,c(1,2,4)],model="mlpe",nr=3,maxit=50,search=4,Runs=5,feature="simp")
L=vector("list",2); L[[1]]=M2; L[[2]]=M1
mgraph(L,graph="REC",xval=0.1,leg=c("mlpe","mr"),main="REC curve")
mgraph(L,graph="DLC",metric="TOLERANCE",xval=0.01,
       leg=c("mlpe","mr"),main="DLC: TOLERANCE plot")
mgraph(M2,graph="IMP",xval=0.01,leg=c("x1","x2"),
       main="sin1reg Input importance",axis=1)
mgraph(M2,graph="VEC",xval=1,main="sin1reg 1-D VEC curve for x1")
mgraph(M2,graph="VEC",xval=1,
       main="sin1reg 1-D VEC curve and histogram for x1",data=sin1reg)

## End(Not run)

### classification example
## Not run:
data(iris)
M1=mining(Species~.,iris,model="rpart",Runs=5) # decision tree (DT)
M2=mining(Species~.,iris,model="ksvm",Runs=5) # support vector machine (SVM)
L=vector("list",2); L[[1]]=M2; L[[2]]=M1
mgraph(M1,graph="ROC",TC=3,leg=-1,baseline=TRUE,Grid=10,main="mr ROC")
mgraph(M1,graph="ROC",TC=3,leg=-1,baseline=TRUE,Grid=10,main="mr ROC",intbar=FALSE)
mgraph(L,graph="ROC",TC=3,leg=c("SVM","DT"),baseline=TRUE,Grid=10,
       main="ROC for virginica")
mgraph(L,graph="LIFT",TC=3,leg=list(pos=c(0.4,0.2),leg=c("SVM","DT")),
       baseline=TRUE,Grid=10,main="LIFT for virginica")

## End(Not run)
```

`mining` *Powerful function that trains and tests a particular fit model under several runs and a given validation method*

Description

Powerful function that trains and tests a particular fit model under several runs and a given validation method. Since there can be a huge number of models, the fitted models are not stored. Yet, several useful statistics (e.g. predictions) are returned.

Usage

```
mining(x, data = NULL, Runs = 1, method = NULL, model = "default",
       task = "default", search = "heuristic", mpar = NULL,
       feature="none", scale = "default", transform = "none",
       debug = FALSE, ...)
```

Arguments

`x` a symbolic description (formula) of the model to be fit. If `x` contains the data, then `data=NULL` (similar to x in `ksvm`, kernlab package).

`data` an optional data frame (columns denote attributes, rows show examples) containing the training data, when using a formula.

`Runs` number of runs used (e.g. 1, 5, 10, 20, 30) `method` a vector with c(*vmethod*,*vpar*,*seed*) or c(*vmethod*,*vpar*,*window*,*increment*), where *vmethod* is:

- `all` – all *NROW* examples are used as both training and test sets (no *vpar* or *seed* is needed).
- `holdout` – standard holdout method. If *vpar*<1 then *NROW**vpar random samples are used for training and the remaining rows are used for testing. Else, then *NROW**vpar random samples are used for testing and the remaining are used for training. For classification tasks (`prob` or `class`) a stratified sampling is assumed (equal to `mode="stratified"` in `holdout`).
- `holdoutrandom` – similar to `holdout` except that assumes always a random sampling (not stratified).
- `holdoutorder` – similar to `holdout` except that instead of a random sampling, the first rows (until the split) are used for training and the remaining ones for testing (equal to `mode="order"` in `holdout`).
- `holdoutinc` – incremental holdout retraining (e.g. used for stock market data). Here, *vpar* is the test size, *window* is the initial window size and *increment* is the number of samples added at each iteration. Note: argument `Runs` is automatically set when this option is used. See also `holdout`.
- `holdoutrol` – rolling holdout retraining (e.g. used for stock market data). Here, *vpar* is the test size, *window* is the window size and *increment* is the number of samples added at each iteration. Note: argument `Runs` is automatically set when this option is used. See also `holdout`.
- `kfold` – K-fold cross-validation method, where *vpar* is the number of folds. For classification tasks (`prob` or `class`) a stratified split is assumed (equal to `mode="stratified"` in `crossvaldata`).
- `kfoldrandom` – similar to `kfold` except that assumes always a random sampling (not stratified).
- `kfoldorder` – similar to `kfold` except that instead of a random sampling, the order of the rows is used to build the folds.

vpar – number used by *vmethod* (optional, if not defined 2/3 for `holdout` and 10 for `kfold` is assumed);
and *seed* (optional, if not defined then `NA` is assumed) is:

- `NA` – random seed is adopted (default R method for generating random numbers);
- a vector of size `Runs` with fixed seed numbers for each Run;
- a number – `set.seed`(*number*) is applied then a vector of seeds (of size Runs) is generated.

`model` See `fit` for details.

`task` See `fit` for details.

`search` See `fit` for details.

`mpar` Only kept for compatibility with previous `rminer` versions, as you should use `search` instead of `mpar`. See `fit` for details.

`feature` See `fit` for more details about `feature="none"`, `"sabs"` or `"sbs"` options.
For the `mining` function, additional options are `feature=`*fmethod*, where *fmethod* can be one of:

- `sens` or `sensg` – compute the 1-D sensitivity analysis input importances (`$sen`), gradient measure.
- `sensv` – compute the 1-D sensitivity analysis input importances (`$sen`), variance measure.
- `sensr` – compute the 1-D sensitivity analysis input importances (`$sen`), range measure.
- `simp`, `simpg` or `s` – equal to `sensg` but also computes the 1-D sensitivity responses (`$sresponses`, useful for `graph="VEC"`).
- `simpv` – equal to `sensv` but also computes the 1-D sensitivity responses (useful for `graph="VEC"`).
- `simpr` – equal to `sensr` but also computes the 1-D sensitivity responses (useful for `graph="VEC"`).

`scale` See `fit` for details.

`transform` See `fit` for details.

`debug` If TRUE shows some information about each run.

`...` See `fit` for details.

Details

Powerful function that trains and tests a particular fit model under several runs and a given validation method (see [Cortez, 2010] for more details).
Several `Runs` are performed. In each run, the same validation method is adopted (e.g. `holdout`) and several relevant statistics are stored. Note: this function can require some computational effort, specially if a large dataset and/or a high number of `Runs` is adopted.

Value

A `list` with the components:

- $time – vector with time elapsed for each run.
- $test – vector list, where each element contains the test (target) results for each run.
- $pred – vector list, where each element contains the predicted results for each test set and each run.
- $error – vector with an error `metric` for each run (the error depends on the `metric` parameter of `mpar`, valid options are explained in `mmetric`).
- $mpar – vector list, where each element contains the fit model mpar parameters (for each run).
- $model – the `model`.
- $task – the `task`.
- $method – the external validation `method`.
- $sen – a matrix with the 1-D sensitivity analysis input importances. The number of rows is `Runs` times *vpar*, if `kfold`, else is `Runs`.
- $sresponses – a vector list with a size equal to the number of attributes (useful for `graph="VEC"`). Each element contains a list with the 1-D sensitivity analysis input responses (`n` – name of the attribute; `l` – number of levels; `x` – attribute values; `y` – 1-D sensitivity responses.
  Important note: sresponses (and "VEC" graphs) are only available if `feature="sabs"` or `"simp"` related (see `feature`).

- $runs – the Runs.
- $attributes – vector list with all attributes (features) selected in each run (and fold if kfold) if a feature selection algorithm is used.
- $feature – the feature.

Note

See also http://www3.dsi.uminho.pt/pcortez/rminer.html

Author(s)

Paulo Cortez http://www3.dsi.uminho.pt/pcortez

References

- To check for more details about rminer and for citation purposes:
  P. Cortez.
  Data Mining with Neural Networks and Support Vector Machines Using the R/rminer Tool.
  In P. Perner (Ed.), Advances in Data Mining - Applications and Theoretical Aspects 10th Industrial Conference on Data Mining (ICDM 2010), Lecture Notes in Artificial Intelligence 6171, pp. 572-583, Berlin, Germany, July, 2010. Springer. ISBN: 978-3-642-14399-1.
  @Springer: http://www.springerlink.com/content/e7u36014r04h0334
  http://www3.dsi.uminho.pt/pcortez/2010-rminer.pdf

See Also

fit, predict.fit, mgraph, mmetric, savemining, holdout and Importance.

Examples

```
### dontrun is used when the execution of the example requires some computational effort.

### simple regression example
x1=rnorm(200,100,20); x2=rnorm(200,100,20)
y=0.7*sin(x1/(25*pi))+0.3*sin(x2/(25*pi))
# mining with an ensemble of neural networks, each fixed with size=2 hidden nodes
# assumes a default holdout (random split) with 2/3 for training and 1/3 for testing:
M=mining(y~x1+x2,Runs=2,model="mlpe",search=2)
print(M)
print(mmetric(M,metric="MAE"))

### more regression examples:
## Not run:
# simple nonlinear regression task; x3 is a random variable and does not influence y:
data(sin1reg)
# 5 runs of an external holdout with 2/3 for training and 1/3 for testing, fixed seed 12345
# feature selection: sabs method
# model selection: 5 searches for size, internal 2-fold cross validation fixed seed 123
#                  with optimization for minimum MAE metric
M=mining(y~.,data=sin1reg,Runs=5,method=c("holdout",2/3,12345),model="mlpe",
       search=list(search="heuristic5",method=c("kfold",2,123),metric="MAE"),feature="sabs")
print(mmetric(M,metric="MAE"))
print(M$mpar)
print("median hidden nodes (size) and number of MLPs (nr):")
print(centralpar(M$mpar))
print("attributes used by the model in each run:")
print(M$attributes)
mgraph(M,graph="RSC",Grid=10,main="sin1 MLPE scatter plot")
mgraph(M,graph="REP",Grid=10,main="sin1 MLPE scatter plot",sort=FALSE)
mgraph(M,graph="REC",Grid=10,main="sin1 MLPE REC")
mgraph(M,graph="IMP",Grid=10,main="input importances",xval=0.1,leg=names(sin1reg))
# average influence of x1 on the model:
mgraph(M,graph="VEC",Grid=10,main="x1 VEC curve",xval=1,leg=names(sin1reg)[1])

## End(Not run)

### regression example with holdout rolling windows:
## Not run:
# simple nonlinear regression task; x3 is a random variable and does not influence y:
data(sin1reg)
# rolling with 20 test samples, training window size of 300 and increment of 50 in each run:
# note that Runs argument is automatically set to 14 in this example:
M=mining(y~.,data=sin1reg,method=c("holdoutrol",20,300,50),
         model="mlpe",search="heuristic",debug=TRUE)

## End(Not run)

### regression example with all rminer models:
## Not run:
# simple nonlinear regression task; x3 is a random variable and does not influence y:
data(sin1reg)
models=c("naive","ctree","rpart","kknn","mlp","mlpe","ksvm","randomForest","mr","mars",
         "cubist","pcr","plsr","cppls","rvm")
for(model in models)
{
 M=mining(y~.,data=sin1reg,method=c("holdout",2/3,12345),model=model)
 cat("model:",model,"MAE:",round(mmetric(M,metric="MAE")$MAE,digits=3),"\n")
}

## End(Not run)

### classification example (task="prob")
## Not run:
data(iris)
# 10 runs of a 3-fold cross validation with fixed seed 123 for generating the 3-fold runs
M=mining(Species~.,iris,Runs=10,method=c("kfold",3,123),model="rpart")
print(mmetric(M,metric="CONF"))
print(mmetric(M,metric="AUC"))
print(meanint(mmetric(M,metric="AUC")))
mgraph(M,graph="ROC",TC=2,baseline=TRUE,Grid=10,leg="Versicolor",
       main="versicolor ROC")
mgraph(M,graph="LIFT",TC=2,baseline=TRUE,Grid=10,leg="Versicolor",
       main="Versicolor ROC")
```

```
M2=mining(Species~.,iris,Runs=10,method=c("kfold",3,123),model="ksvm")
L=vector("list",2)
L[[1]]=M;L[[2]]=M2
mgraph(L,graph="ROC",TC=2,baseline=TRUE,Grid=10,leg=c("DT","SVM"),main="ROC")

## End(Not run)

### other classification examples
## Not run:
### 1st example:
data(iris)
# 2 runs of an external 2-fold validation, random seed
# model selection: SVM model with rbfdot kernel, automatic search for sigma,
#                  internal 3-fold validation, random seed, minimum "AUC" is assumed
# feature selection: none, "s" is used only to store input importance values
M=mining(Species~.,data=iris,Runs=2,method=c("kfold",2,NA),model="ksvm",
         search=list(search="heuristic",method=c("kfold",3)),feature="s")

print(mmetric(M,metric="AUC",TC=2))
mgraph(M,graph="ROC",TC=2,baseline=TRUE,Grid=10,leg="SVM",main="ROC",intbar=FALSE)
mgraph(M,graph="IMP",TC=2,Grid=10,main="input importances",xval=0.1,
leg=names(iris),axis=1)
mgraph(M,graph="VEC",TC=2,Grid=10,main="Petal.Width VEC curve",
data=iris,xval=4)
### 2nd example, ordered kfold, k-nearest neigbor:
M=mining(Species~.,iris,Runs=1,method=c("kfoldo",3),model="knn")
# confusion matrix:
print(mmetric(M,metric="CONF"))

### 3rd example, use of all rminer models:
models=c("naive","ctree","rpart","kknn","mlp","mlpe","ksvm","randomForest","bagging",
         "boosting","lda","multinom","naiveBayes","qda")
models="naiveBayes"
for(model in models)
{
 M=mining(Species~.,iris,Runs=1,method=c("kfold",3,123),model=model)
 cat("model:",model,"ACC:",round(mmetric(M,metric="ACC")$ACC,digits=1),"\n")
}

## End(Not run)

### for more fitting examples check the help of function fit: ?fit
```

mmetric *Compute classification or regression error metrics.*

Description

Compute classification or regression error metrics.

Usage

```
mmetric(y, x = NULL, metric, D = 0.5, TC = -1, val = NULL, aggregate = "no")
```

Arguments

`y`
: if there are predictions (`!is.null(x)`), `y` should be a numeric vector or factor with the target desired responses (or output values).
Else, `y` should be a list returned by the `mining` function.

`x`
: the predictions (should be a numeric vector if `task="reg"`, matrix if `task="prob"` or factor if `task="class"` (used if `y` is not a list).

`metric`
: a R function or a character.
Note: if a R function, then it should be set to provide lower values for better models if the intention is to be used within the `search` argument of `fit` and `mining` (i.e., "<" meaning).
Valid character options are (">" means "better" if higher value; "<" means "better" if lower value):

- `ALL` – returns all classification or regression metrics (context dependent, multi-metric).
- if vector – returns all metrics included in the vector, vector elements can be any of the options below (multi-metric).
- `CONF` – confusion matrix (classification, matrix).
- `ACC` – classification accuracy rate (classification, ">", [0-%100]).
- `CE` – classification error or misclassification error rate (classification, "<", [0-%100]).
- `MAEO` – mean absolute error for ordinal classification (classification, "<", [0-Inf[).
- `MSEO` – mean squared error for ordinal classification (classification, "<", [0-Inf[).
- `KENDALL` – Kendalls's coefficient for ordinal classification or (mean if) ranking (classification, ">", [-1;-1]). Note: if ranking, `y` is a matrix and mean metric is computed.
- `SPEARMAN` – Mean Spearman's rho coefficient for ranking (classification, ">", [-1;-1]). Note: if ranking, `y` is a matrix and mean metric is computed.
- `BER` – balanced error rate (classification, "<", [0-%100]).
- `KAPPA` – kappa index (classification, "<", [0-%100]).
- `CRAMERV` – Cramer's V (classification, ">", [0,1.0]).
- `ACCLASS` – classification accuracy rate per class (classification, ">", [0-%100]).
- `TPR` – true positive rate, sensitivity or recall (classification, ">", [0-%100]).
- `TNR` – true negative rate or specificity (classification, ">", [0-%100]).
- `PRECISION` – precision (classification, ">", [0-%100]).
- `F1` – F1 score (classification, ">", [0-%100]).
- `MCC` – Matthews correlation coefficient (classification, ">", [-1,1]).
- `BRIER` – overall Brier score (classification "prob", "<", [0,1.0]).

- BRIERCLASS – Brier score per class (classification "prob", "<", [0,1.0]).
- ROC – Receiver Operating Characteristic curve (classification "prob", list with several components).
- AUC – overall area under the curve (of ROC curve, classification "prob", ">", domain values: [0,1.0]).
- AUCCLASS – area under the curve per class (of ROC curve, classification "prob", ">", domain values: [0,1.0]).
- NAUC – normalized AUC (given a fixed val=FPR, classification "prob", ">", [0,1.0]).
- TPRATFPR – the TPR (given a fixed val=FPR, classification "prob", ">", [0,1.0]).
- LIFT – accumulative percent of responses captured (LIFT accumulative curve, classification "prob", list with several components).
- ALIFT – area of the accumulative percent of responses captured (LIFT accumulative curve, classification "prob", ">", [0,1.0]).
- NALIFT – normalized ALIFT (given a fixed val=percentage of examples, classification "prob", ">", [0,1.0]).
- ALIFTATPERC – ALIFT value (given a fixed val=percentage of examples, classification "prob", ">", [0,1.0]).
- SAE – sum absolute error/deviation (regression, "<", [0,Inf[).
- MAE – mean absolute error (regression, "<", [0,Inf[).
- MdAE – median absolute error (regression, "<", [0,Inf[).
- GMAE – geometric mean absolute error (regression, "<", [0,Inf[).
- MaxAE – maximum absolute error (regression, "<", [0,Inf[).
- RAE – relative absolute error (regression, "<", [0%,Inf[).
- SSE – sum squared error (regression, "<", [0,Inf[).
- MSE – mean squared error (regression, "<", [0,Inf[).
- MdSE – median squared error (regression, "<", [0,Inf[).
- RMSE – root mean squared error (regression, "<", [0,Inf[).
- GMSE – geometric mean squared error (regression, "<", [0,Inf[).
- HRMSE – Heteroscedasticity consistent root mean squared error (regression, "<", [0,Inf[).
- RSE – relative squared error (regression, "<", [0%,Inf[).
- RRSE – root relative squared error (regression, "<", [0%,Inf[).
- ME – mean error (regression, "<", [0,Inf[).
- SMinkowski3 – sum of Minkowski loss function (q=3, heavier penalty for large errors when compared with SSE, regression, "<", [0%,Inf[).
- MMinkowski3 – mean of Minkowski loss function (q=3, heavier penalty for large errors when compared with SSE, regression, "<", [0%,Inf[).
- MdMinkowski3 – median of Minkowski loss function (q=3, heavier penalty for large errors when compared with SSE, regression, "<", [0%,Inf[).
- COR – correlation (regression, ">", [-1,1]).
- q2 – =1-correlation^2 test error metric, as used by M.J. Embrechts (regression, "<", [0,1.0]).

- R2 – coefficient of determination R^2 (regression, ">", squared pearson correlation coefficient: [0,1]).
- R22 – 2nd variant of coefficient of determination R^2 (regression, ">", most general definition that however can lead to negative values:]-Inf,1]. In previous rminer versions, this variant was known as "R2").
- Q2 – R^2/SD test error metric, as used by M.J. Embrechts (regression, "<", [0,Inf[).
- REC – Regression Error Characteristic curve (regression, list with several components).
- NAREC – normalized REC area (given a fixed val=tolerance, regression, ">", [0,1.0]).
- TOLERANCE – the tolerance (y-axis value) of a REC curve (given a fixed val=tolerance, regression, ">", [0,1.0]).
- MAPE – Mean Absolute Percentage mmetric forecasting metric (regression, "<", [0%,Inf[).
- MdAPE – Mean Absolute Percentage mmetric forecasting metric (regression, "<"), [0%,Inf[).
- RMSPE – Root Mean Square Percentage mmetric forecasting metric (regression, "<", [0%,Inf[).
- RMdSPE – Root Median Square Percentage mmetric forecasting metric (regression, "<", [0%,Inf[).
- SMAPE – Symmetric Mean Absolute Percentage mmetric forecasting metric (regression, "<", [0%,200%]).
- SMdAPE – Symmetric Median Absolute Percentage mmetric forecasting metric (regression, "<", [0%,200%]).
- MRAE – Mean Relative Absolute mmetric forecasting metric (val should contain the last in-sample/training data value (for random walk) or full benchmark time series related with out-of-sample values, regression, "<", [0,Inf[).
- MdRAE – Median Relative Absolute mmetric forecasting metric (val should contain the last in-sample/training data value (for random walk) or full benchmark time series, regression, "<", [0,Inf[).
- GMRAE – Geometric Mean Relative Absoluate mmetric forecasting metric (val should contain the last in-sample/training data value (for random walk) or full benchmark time series, regression, "<", [0,Inf[).
- THEILSU2 – Theils'U2 forecasting metric (val should contain the last in-sample/training data value (for random walk) or full benchmark time series, regression, "<", [0,Inf[).
- MASE – MASE forecasting metric (val should contain the time series in-samples or training data, regression, "<", [0,Inf[).

D

decision threshold (for task="prob", probabilistic classification) within [0,1]. The class is TRUE if *prob>D*.

TC

target class index or vector of indexes (for multi-class classification class) within 1,...,*Nc*, where *Nc* is the number of classes:<cr>

- if TC==-1 (the default value), then it is assumed:

  - if `metric` is "CONF" – `D` is ignored and highest probability class is assumed (if `TC>0`, the metric is computed for positive `TC` class and `D` is used).
  - if `metric` is "ACC", "CE", "BER", "KAPPA", "CRAMERV", "BRIER", or "AUC" – the global metric (for all classes) is computed (if `TC>0`, the metric is computed for positive `TC` class).
  - if `metric` is "ACCLASS", "TPR", "TNR", "Precision", "F1", "MCC", "ROC", "BRIERCLASS", "AUCCLASS" – it returns one result per class (if `TC>0`, it returns negative (e.g. "TPR1") and positive (TC, e.g. "TPR2") result).
  - if `metric` is "NAUC", "TPRATFPR", "LIFT", "ALIFT", "NALIFT" or "ALIFTATPERC" – TC is set to the index of the last class.

`val` auxiliary value:

- when two or more metrics need different `val` values, then `val` should be a vector list, see example.
- if numeric or vector – check the `metric` argument for specific details of each metric val meaning.

`aggregate` character with type of aggregation performed when `y` is a `mining` list. Valid options are:

- no – returns all metrics for all `mining` runs. If `metric` includes "CONF", "ROC", "LIFT" or "REC", it returns a vector list, else if `metric` includes a single metric, it returns a vector; else it retuns a data.frame (runs x metrics).
- sum – sums all run results.
- mean – averages all run results.
- note: both "sum" and "mean" only work if only `metric=="CONF"` is used or if `metric` does not contain "ROC", "LIFT" or "REC".

Details

Compute classification or regression error metrics:

- `mmetric` – compute one or more classification/regression metrics given y and x OR a mining list.
- `metrics` – deprecated function, same as `mmetric(x,y,metric="ALL")`, included here just for compatability purposes but will be removed from the package.

Value

Returns the computed error metric(s):

- one value if only one `metric` is requested (and `y` is not a mining list);
- named vector if 2 or more elements are requested in `metric` (and `y` is not a mining list);
- list if there is a "CONF", "ROC", "LIFT" or "REC" request on `metric` (other metrics are stored in field `$res`, and `y` is not a mining list).
- if `y` is a mining list then there can be several runs, thus:

- a vector list of size `y$runs` is returned if `metric` includes "CONF", "ROC", "LIFT" or "REC" and `aggregate="no"`;
- a data.frame is returned if `aggregate="no"` and `metric` does not include "CONF", "ROC", "LIFT" or "REC";
- a table is returned if `aggregate="sum" or "mean"` and `metric="CONF"`;
- a vector or numeric value is returned if `aggregate="sum" or "mean"` and `metric` is not "CONF".

Note

See also `http://www3.dsi.uminho.pt/pcortez/rminer.html`

Author(s)

Paulo Cortez `http://www3.dsi.uminho.pt/pcortez`

References

- To check for more details about rminer and for citation purposes:
  P. Cortez.
  Data Mining with Neural Networks and Support Vector Machines Using the R/rminer Tool.
  In P. Perner (Ed.), Advances in Data Mining - Applications and Theoretical Aspects 10th Industrial Conference on Data Mining (ICDM 2010), Lecture Notes in Artificial Intelligence 6171, pp. 572-583, Berlin, Germany, July, 2010. Springer. ISBN: 978-3-642-14399-1.
  @Springer: `http://www.springerlink.com/content/e7u36014r04h0334`
  `http://www3.dsi.uminho.pt/pcortez/2010-rminer.pdf`

- About the Brier and Global AUC scores:
  A. Silva, P. Cortez, M.F. Santos, L. Gomes and J. Neves.
  Rating Organ Failure via Adverse Events using Data Mining in the Intensive Care Unit.
  In Artificial Intelligence in Medicine, Elsevier, 43 (3): 179-193, 2008.
  `http://dx.doi.org/10.1016/j.artmed.2008.03.010`

- About the classification and regression metrics:
  I. Witten and E. Frank.
  Data Mining: Practical machine learning tools and techniques.
  Morgan Kaufmann, 2005.
- About the forecasting metrics:
  R. Hyndman and A. Koehler
  Another look at measures of forecast accuracy.
  In International Journal of Forecasting, 22(4):679-688, 2006.

- About the ordinal classification metrics:
  J.S. Cardoso and R. Sousa.
  Measuring the Performance of Ordinal Classification.
  In International Journal of Pattern Recognition and Artificial Intelligence, 25(8):1173-1195, 2011.

See Also

fit, predict.fit, mining, mgraph, savemining and Importance.

Examples

```
### regression examples: y - desired values; x - predictions
y=c(95.01,96.1,97.2,98.0,99.3,99.7);x=95:100
print(mmetric(y,x,"ALL"))
print(mmetric(y,x,"MAE"))
m=mmetric(y,x,c("MAE","RMSE","RAE","RSE"))
print(m)
cat(names(m)[3],"=",round(m[3],digit=2),"\n",sep="")
print(mmetric(y,x,c("COR","R2","Q2")))
print(mmetric(y,x,c("TOLERANCE","NAREC"),val=0.5))
print(mmetric(y,x,"THEILSU2",val=94.1)) # val = 1-ahead random walk, c(y,94.1), same as below
print(mmetric(y,x,"THEILSU2",val=c(94.1,y[1:5]))) # val = 1-ahead random walk (previous y values)
print(mmetric(y,x,"MASE",val=c(88.1,89.9,93.2,94.1))) # val = in-samples
val=vector("list",length=4)
val[[2]]=0.5;val[[3]]=94.1;val[[4]]=c(88.1,89.9,93.2,94.1)
print(mmetric(y,x,c("MAE","NAREC","THEILSU2","MASE"),val=val))
# user defined error function example:
# myerror = number of samples with absolute error above 0.1% of y:
myerror=function(y,x){return (sum(abs(y-x)>(0.001*y)))}
print(mmetric(y,x,metric=myerror))
# example that returns a list since "REC" is included:
print(mmetric(y,x,c("MAE","REC","TOLERANCE"),val=1))

### pure binary classification
y=factor(c("a","a","a","a","b","b","b","b"))
x=factor(c("a","a","b","a","b","a","b","a"))
print(mmetric(y,x,"CONF")$conf)
print(mmetric(y,x,"ALL"))
print(mmetric(y,x,metric=c("ACC","TPR","ACCLASS")))

### probabilities binary classification
y=factor(c("a","a","a","a","b","b","b","b"))
px=matrix(nrow=8,ncol=2)
px[,1]=c(1.0,0.9,0.8,0.7,0.6,0.5,0.4,0.3)
px[,2]=1-px[,1]
print(px)
print(mmetric(y,px,"CONF")$conf)
print(mmetric(y,px,"CONF",D=0.5,TC=2)$conf)
print(mmetric(y,px,"CONF",D=0.3,TC=2)$conf)
print(mmetric(y,px,metric="ALL",D=0.3,TC=2))
print(mmetric(y,px,metric=c("ACC","AUC","AUCCLASS","BRIER","BRIERCLASS","CE"),D=0.3,TC=2))

### pure multi-class classification
y=factor(c("a","a","b","b","c","c"))
x=factor(c("a","a","b","c","b","c"))
print(mmetric(y,x,metric="CONF")$conf)
print(mmetric(y,x,metric="CONF",TC=-1)$conf)
print(mmetric(y,x,metric="CONF",TC=1)$conf)
print(mmetric(y,x,metric="ALL"))
print(mmetric(y,x,metric=c("ACC","ACCLASS","KAPPA")))
print(mmetric(y,x,metric=c("ACC","ACCLASS","KAPPA"),TC=1))

### probabilities multi-class
y=factor(c("a","a","b","b","c","c"))
px=matrix(nrow=6,ncol=3)
px[,1]=c(1.0,0.7,0.5,0.3,0.1,0.7)
px[,2]=c(0.0,0.2,0.4,0.7,0.3,0.2)
px[,3]=1-px[,1]-px[,2]
print(px)
print(mmetric(y,px,metric=c("AUC","AUCCLASS","NAUC"),TC=-1,val=0.1))
print(mmetric(y,px,metric=c("AUC","NAUC"),TC=3,val=0.1))
print(mmetric(y,px,metric=c("ACC","ACCLASS"),TC=-1))
print(mmetric(y,px,metric=c("CONF"),TC=3,D=0.5)$conf)
print(mmetric(y,px,metric=c("ACCLASS"),TC=3,D=0.5))
print(mmetric(y,px,metric=c("CONF"),TC=3,D=0.7)$conf)
print(mmetric(y,px,metric=c("ACCLASS"),TC=3,D=0.7))

### ordinal multi-class (example in Ricardo Sousa PhD thesis 2012)
y=ordered(c(rep("a",4),rep("b",6),rep("d",3)),levels=c("a","b","c","d"))
x=ordered(c(rep("c",(4+6)),rep("d",3)),levels=c("a","b","c","d"))
print(mmetric(y,x,metric="CONF")$conf)
print(mmetric(y,x,metric=c("CE","MAEO","MSEO","KENDALL")))
# note: only y needs to be ordered
x=factor(c(rep("b",4),rep("a",6),rep("d",3)),levels=c("a","b","c","d"))
print(mmetric(y,x,metric="CONF")$conf)
print(mmetric(y,x,metric=c("CE","MAEO","MSEO","KENDALL")))

### ranking (multi-class)
y=matrix(nrow=1,ncol=12);x=y
# http://www.youtube.com/watch?v=D56dvoVrBBE
y[1,]=1:12
x[1,]=c(2,1,4,3,6,5,8,7,10,9,12,11)
print(mmetric(y,x,metric="KENDALL"))
print(mmetric(y,x,metric="ALL"))

y=matrix(nrow=2,ncol=7);x=y
y[1,]=c(2,6,5,4,3,7,1)
y[2,]=7:1
x[1,]=1:7
x[2,]=1:7
print(mmetric(y,x,metric="ALL"))

### mining, several runs, prob multi-class
## Not run:
data(iris)
M=mining(Species~.,iris,model="rpart",Runs=2)
R=mmetric(M,metric="CONF",aggregate="no")
print(R[[1]]$conf)
print(R[[2]]$conf)
print(mmetric(M,metric="CONF",aggregate="mean"))
print(mmetric(M,metric="CONF",aggregate="sum"))
print(mmetric(M,metric=c("ACC","ACCLASS"),aggregate="no"))
print(mmetric(M,metric=c("ACC","ACCLASS"),aggregate="mean"))
print(mmetric(M,metric="ALL",aggregate="no"))
print(mmetric(M,metric="ALL",aggregate="mean"))

## End(Not run)

### mining, several runs, regression
## Not run:
data(sin1reg)
S=sample(1:nrow(sin1reg),40)
M=mining(y~.,data=sin1reg[S,],model="ksvm",search=2^3,Runs=10)
R=mmetric(M,metric="MAE")
print(mmetric(M,metric="MAE",aggregate="mean"))
miR=meanint(R) # mean and t-student confidence intervals
cat("MAE=",round(miR$mean,digits=2),"+-",round(miR$int,digits=2),"\n")
print(mmetric(M,metric=c("MAE","RMSE")))
print(mmetric(M,metric=c("MAE","RMSE"),aggregate="mean"))
R=mmetric(M,metric="REC",aggregate="no")
print(R[[1]]$rec)
print(mmetric(M,metric=c("TOLERANCE","NAREC"),val=0.2))
print(mmetric(M,metric=c("TOLERANCE","NAREC"),val=0.2,aggregate="mean"))

## End(Not run)
```

predict.fit *predict method for fit objects (rminer)*

Description

predict method for fit objects (rminer)

Arguments

`object`	a model object created by `fit`
`newdata`	a data frame or matrix containing new data

Details

Returns predictions for a fit model. Note: the `...` optional argument is currently only used by `cubist` model (see example).

Value

If `task` is `prob` returns a matrix, where each column is the class probability.
If `task` is `class` returns a factor.
If `task` is `reg` returns a numeric vector.

Methods

`signature(object = "model")` describe this method here

References

- To check for more details about rminer and for citation purposes:
  P. Cortez.
  Data Mining with Neural Networks and Support Vector Machines Using the R/rminer Tool.
  In P. Perner (Ed.), Advances in Data Mining - Applications and Theoretical Aspects 10th Industrial Conference on Data Mining (ICDM 2010), Lecture Notes in Artificial Intelligence 6171, pp. 572-583, Berlin, Germany, July, 2010. Springer. ISBN: 978-3-642-14399-1.
  @Springer: http://www.springerlink.com/content/e7u36014r04h0334
  http://www3.dsi.uminho.pt/pcortez/2010-rminer.pdf

See Also

`fit`, `mining`, `mgraph`, `mmetric`, `savemining`, `CasesSeries`, `lforecast` and `Importance`.

Examples

```
### simple classification example with logistic regression
data(iris)
M=fit(Species~.,iris,model="lr")
P=predict(M,iris)
print(mmetric(iris$Species,P,"CONF")) # confusion matrix

### simple regression example
data(sa_ssin)
H=holdout(sa_ssin$y,ratio=0.5,seed=12345)
Y=sa_ssin[H$ts,]$y # desired test set
# fit multiple regression on training data (half of samples)
M=fit(y~.,sa_ssin[H$tr,],model="mr") # multiple regression
P1=predict(M,sa_ssin[H$ts,]) # predictions on test set
print(mmetric(Y,P1,"MAE")) # mean absolute error

### fit cubist model
M=fit(y~.,sa_ssin[H$tr,],model="cubist") #
P2=predict(M,sa_ssin[H$ts,],neighbors=3) #
print(mmetric(Y,P2,"MAE")) # mean absolute error
P3=predict(M,sa_ssin[H$ts,],neighbors=7) #
print(mmetric(Y,P3,"MAE")) # mean absolute error

### check fit for more examples
```

savemining *Load/save into a file the result of a fit (model) or mining functions.*

Description

Load/save into a file the result of a fit (model) or mining functions.

Usage

```
savemining(mmm_mining, file, ascii = TRUE)
```

Arguments

mmm_mining	the list object that is returned by the mining function.
file	filename that should include an extension
ascii	if TRUE then ascii format is used to store the file (larger file size), else a binary format is used.

Details

Very simple functions that do what their names say. Additional usages are:
`loadmining(file)`
`savemodel(MM_model,file,ascii=FALSE)`
`loadmodel(file)`

Value

`loadmining` returns a `mining` mining list, while `loadmodel` returns a `model` object (from `fit`).

Author(s)

Paulo Cortez http://www3.dsi.uminho.pt/pcortez

References

See `fit`.

See Also

`fit`, `predict.fit`, `mining`, `mgraph`, `mmetric`, `savemining`, `Importance`.

Examples

```
data(iris)
M=fit(Species~.,iris,model="rpart")
savemodel(M,"iris.model") # saves to file
M=NULL # cleans M
M=loadmodel("iris.model") # load from file
print(M)
```

`sa_fri1` *Synthetic regression and classification datasets for measuring input importance of supervised learning models*

Description

5 Synthetic regression (sa_fri1, sa_ssin, sa_psin, sa_int2, sa_tree) and 4 classification (sa_ssin_2, sa_ssin_n2p, sa_int2_3c, sa_int2_8p) datasets for measuring input importance of supervised learning models

Usage

```
data(sa_fri1)
```

Format

A data frame with 1000 observations on the following variables.

x**n** input (numeric or factor, depends on the dataset)

y output target (numeric or factor, depends on the dataset)

Details

Check reference or source for full details

Source

See references

References

- To cite the Importance function, sensitivity analysis methods or synthetic datasets, please use:
  P. Cortez and M.J. Embrechts.
  Using Sensitivity Analysis and Visualization Techniques to Open Black Box Data Mining Models.
  In Information Sciences, Elsevier, 225:1-17, March 2013.
  http://dx.doi.org/10.1016/j.ins.2012.10.039

Examples

```
data(sa_ssin)
print(summary(sa_ssin))
## Not run: plot(sa_ssin$x1,sa_ssin$y)
```

`sin1reg` *sin1 regression dataset*

Description

Simple synthetic dataset with 1000 points, where y=0.7*sin(pi*x1/2000)+0.3*sin(pi*x2/2000)

Usage

```
data(sin1reg)
```

Format

The format is: chr "sin1reg"

Details

Simple synthetic dataset with 1000 points, where y=0.7*sin(pi*x1/2000)+0.3*sin(pi*x2/2000)

Source

See references

References

- To cite the Importance function, sensitivity analysis methods or synthetic datasets, please use:
  P. Cortez and M.J. Embrechts.
  Using Sensitivity Analysis and Visualization Techniques to Open Black Box Data Mining Models.
  In Information Sciences, Elsevier, 225:1-17, March 2013.
  http://dx.doi.org/10.1016/j.ins.2012.10.039

Examples

```
data(sin1reg)
print(summary(sin1reg))
```

vecplot *VEC plot function (to use in conjunction with Importance function).*

Description

VEC plot function (to use in conjunction with Importance function).

Usage

```
vecplot(I, graph = "VEC", leg = NULL, xval = 1, sort = FALSE, data = NULL,
digits = c(1, 1), TC = 1, intbar = NULL, lty = 1, pch = 19, col = NULL,
datacol = NULL, main = "", main2 = "", Grid = 0,
xlab = "", ylab = "", zlab = "",
levels = NULL, levels2 = NULL, showlevels = FALSE,
screen = list(z = 40, x = -60), zoom = 1, cex = 1)
```

Arguments

I the output list of the Importance function.

graph type of VEC graph:

- VEC – 1-D VEC curve;
- VECB – 1-D VEC curve with box plots (only valid for SA methods: "DSA", "MSA");
- VEC3 – 2-D VEC surface;
- VECC – 2-D VEC contour;

leg see mgraph

xval the attribute input index (e.g. 1), only used if graph="VEC" or (graph="VEC3" or "VECC" and length(interactions)=1, see Importance). if a vector, then several VEC curves are plotted (in this case, x-axis is scaled).

sort if factor inputs are sorted:

- increasing – sorts the first attribute (if factor) according to the response values, increasing order;
- decreasing – similar to increasing but uses reverse order;
- TRUE – similar to increasing;
- increasing2 – sorts the second attribute (for graph="VEC3" or "VECC", if factor, according to the response values), increasing order;
- decreasing2 – similar to increasing2 but uses reverse order;
- FALSE – no sort is used;

data see mgraph

digits see mgraph

TC see mgraph

intbar see mgraph

`lty` see `mgraph`

`pch` point type for the `graph="VEC"` curve, can be a vector if there are several VEC curve plots

`col` color (e.g. "black", "grayrange", "white")

`datacol` color of the data histogram for `graph="VEC"`

`main` see `mgraph`

`main2` key title for `graph="VECC"`

`Grid` see `mgraph`

`xlab` x-axis label

`ylab` y-axis label

`zlab` z-axis label

`levels` if x1 is factor you can choose the order of the levels to this argument

`levels2` if x2 is factor you can choose the order of the levels to this argument

`showlevels` if you want to show the factor levels in x1 or x2 axis in `graph="VEC3"`:

- `FALSE` or `TRUE` – do not (do) show the levels in x1, x2 and z axis for factor variables;
- vector with 3 logical values – if you want to show the levels in each of the x1, x2 or z axis for factor variables (e.g. `c(FALSE,FALSE,TRUE)` only shows for z-axis).

`screen` select the perspective angle of the `VEC3` graph:

- `x` – assumes `list(z=0,x=-90,y=0)`;
- `X` – assumes `list(x=-75)`;
- `y` – assumes `list(z=0,x=-90,y=-90)`;
- `Y` – assumes `list(z=10,x=-90,y=-90)`;
- `z` – assumes `list(z=0,x=0,y=0)`;
- `xy` – assumes `list(z=10,x=-90,y=-45)`;
- else you need to specify a list with z, x an y angles, see `wireframe`

`zoom` zoom of the wireframe (`graph="VEC3"`)

`cex` label font size

Details

For examples and references check: `Importance`

Value

A VEC curve/surface/contour plot.

Author(s)

Paulo Cortez `http://www3.dsi.uminho.pt/pcortez`

References

- To cite the Importance function or sensitivity analysis method, please use:

  P. Cortez and M.J. Embrechts.
  Using Sensitivity Analysis and Visualization Techniques to Open Black Box Data Mining Models.
  In Information Sciences, Elsevier, 225:1-17, March 2013.

  http://dx.doi.org/10.1016/j.ins.2012.10.039

See Also

Importance

Index