Package 'performanceEstimation'

October 14, 2022

Type Package

Title An Infra-Structure for Performance Estimation of Predictive Models

Description An infra-structure for estimating the predictive performance of predictive models. In this context, it can also be used to compare and/or select among different alternative ways of solving one or more predictive tasks. The main goal of the package is to provide a generic infra-structure to estimate the values of different metrics of predictive performance using different estimation procedures. These estimation tasks can be applied to any solutions (workflows) to the predictive tasks. The package provides easy to use standard workflows that allow the usage of any available R modeling algorithm together with some pre-defined data pre-processing steps and also prediction post-processing methods. It also provides means for addressing issues related with the statistical significance of the observed differences.

Version 1.1.0

Depends R(>= 3.0), methods

Imports ggplot2 (>= 0.9.3), parallelMap (>= 1.3), parallel, tidyr (>= 0.4.1), dplyr (>= 0.4.3)

Date 2016-10-12

URL https://github.com/ltorgo/performanceEstimation

BugReports https://github.com/ltorgo/performanceEstimation/issues

License GPL (>= 2)

LazyLoad yes

LazyData yes

Suggests knitr, rmarkdown, devtools, e1071, DMwR, randomForest, quantmod, nnet, mlbench, MASS

NeedsCompilation no

Author Luis Torgo [aut, cre]

Maintainer Luis Torgo <ltorgo@dcc.fc.up.pt>

Repository CRAN

Date/Publication 2016-10-13 20:37:05

R topics documented:

`bootEstimates` *Performance estimation using (e0 or .632) bootstrap*

Description

This function obtains boostrap estimates of performance metrics for a given predictive task and method to solve it (i.e. a workflow). The function is general in the sense that the workflow function that the user provides as the solution to the task, can implement or call whatever modeling technique the user wants.

The function implements both e0 boostrap estimates as well as .632 boostrap. The selection of the type of boostrap is done through the `estTask` argument (check the help page of `Bootstrap`).

Please note that most of the times you will not call this function directly, though there is nothing wrong in doing it, but instead you will use the function `performanceEstimation`, that allows you to carry out performance estimation for multiple workflows on multiple tasks, using some estimation method like for instance boostrap. Still, when you simply want to have the boostrap estimate for one workflow on one task, you may use this function directly.

Usage

```
bootEstimates(wf,task,estTask,cluster)
```

Arguments

`wf`	an object of the class `Workflow` representing the modeling approach to be evaluated on a certain task.
`task`	an object of the class `PredTask` representing the prediction task to be used in the evaluation.
`estTask`	an object of the class `EstimationTask` indicating the metrics to be estimated and the boostrap settings to use.
`cluster`	an optional parameter that can either be `TRUE` or a `cluster`. In case of `TRUE` the function will run in parallel and will internally setup the parallel back-end (defaulting to using half of the cores in your local machine). You may also setup outside your parallel back-end (c.f. `makeCluster`) and then pass the resulting `cluster` object to this function using this parameter. In case no value is provided for this parameter the function will run sequentially.

Details

The idea of this function is to carry out a bootstrap experiment with the goal of obtaining reliable estimates of the predictive performance of a certain modeling approach (denoted here as a *workflow*) on a given predictive task. Two types of bootstrap estimates are implemented: i) e0 bootstrap and ii) .632 bootstrap. Bootstrap estimates are obtained by averaging over a set of k scores each obtained in the following way: i) draw a random sample with replacement with the same size as the original data set; ii) obtain a model with this sample; iii) test it and obtain the estimates for this run on the observations of the original data set that were not used in the sample obtained in step i). This process is repeated k times and the average scores are the bootstrap estimates. The main difference between e0 and .632 bootstrap is the fact that the latter tries to integrate the e0 estimate with the resubstitution estimate, i.e. when the model is learned and tested on the full available data sample.

Parallel execution of the estimation experiment is only recommended for minimally large data sets otherwise you may actually increase the computation time due to communication costs between the processes.

Value

The result of the function is an object of class `EstimationResults`.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

`Bootstrap`, `Workflow`, `standardWF`, `PredTask`, `EstimationTask`, `performanceEstimation`, `hldEstimates`, `loocvEstimates`, `cvEstimates`, `mcEstimates`, `EstimationResults`

Examples

```
## Not run:

## Estimating the MSE of a SVM variant on the
##  swiss data, using 50 repetitions of .632 bootstrap
library(e1071)
data(swiss)

## running the estimation experiment
res <- bootEstimates(
  Workflow(wfID="svmC10G01",
           learner="svm",learner.pars=list(cost=10,gamma=0.1)
          ),
  PredTask(Infant.Mortality ~ .,swiss),
  EstimationTask("mse",method=Bootstrap(type=".632",nReps=50))
  )
```

```
## Check a summary of the results
summary(res)


## End(Not run)
```

`Bootstrap-class` *Class "Bootstrap"*

Description

This class of objects contains the information describing a bootstrap experiment, i.e. its settings.

Objects from the Class

Objects can be created by calls of the form `Bootstrap(...)` providing the values for the class slots. The objects contain information on the type of boostrap, the number of repetitions, the random number generator seed and *optionally* the concrete data splits to use on each iteration of the boostrap experiment. Note that most of the times you will not supply these data splits as the boostrap routines in this infra-structure will take care of building them. Still, this allows you to replicate some experiment carried out with specific train/test splits.

Slots

`type`**:** Object of class `character` indicating the type of boostrap estimates to use: "e0" (default) or ".632".

`nReps`**:** Object of class `numeric` indicating the number of repetitions of the bootstrap experiment (defaulting to 200).

`seed`**:** Object of class `numeric` with the random number generator seed (defaulting to 1234).

`dataSplits`**:** Object of class `list` containing the data splits to use on each bootstrap repetition. Each element should be a list with two components: `test` and `train`, on this order. Each of these is a vector with the row ids to use as test and train sets of each repetition of the bootstrap experiment.

Extends

Class `EstCommon`, directly. Class `EstimationMethod`, directly.

Methods

show `signature(object = "Bootstrap")`: method used to show the contents of a `Bootstrap` object.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

MonteCarlo, LOOCV, CV, Holdout, EstimationMethod, EstimationTask

Examples

```
showClass("Bootstrap")

s <- Bootstrap(type=".632",nReps=400)
s

## Small example illustrating the format of user supplied data splits
s2 <- Bootstrap(dataSplits=list(list(test=sample(1:150,50),train=sample(1:150,50)),
                                list(test=sample(1:150,50),train=sample(1:150,50)),
                                list(test=sample(1:150,50),train=sample(1:150,50))
                               ))
s2
s2@dataSplits
```

CDdiagram.BD *CD diagrams for the post-hoc Boferroni-Dunn test*

Description

This function obtains a Critical Difference (CD) diagram for the post-hoc Bonferroni-Dunn test along the lines defined by Demsar (2006). These diagrams provide an interesting visualization of the statistical significance of the observed paired differences between a set of workflows and a selected baseline wrokflow. They allow us to compare a set of alternative workflows against this baseline and answer the question whether the differences are statisticall y significant.

Usage

```
CDdiagram.BD(r, metric = names(r)[1])
```

Arguments

r	A list resulting from a call to pairedComparisons
metric	The metric for which the CD diagram will be obtained (defaults to the first metric of the comparison).

Details

Critical Difference (CD) diagrams are interesting sucint visualizations of the results of a Bonferroni-Dunn post-hoc test that is designed to check the statistical significance between the differences in average rank of a set of workflows against a baseline workflow, on a set of predictive tasks.

In the resulting graph each workflow is represented by a colored line. The X axis where the lines end represents the average rank position of the respective workflow across all tasks. The null hypothesis is that the average rank of a baseline workflow does not differ with statistical significance (at some confidence level defined in the call to `pairedComparisons` that creates the object used to obtain these graphs) from the average ranks of a set of alternative workflows. An horizontal line connects the baseline workflow with the alternative workflows for which we cannot reject this hypothesis. This means that only the alternative workflows that are not connect with the baseline can be considered as having an average rank that is different from the one of the baseline with statistical significance. To help spotting these differences the name of the baseline workflow is shown in bold, and the names of the alternative workflows whose difference is significant are shown in italics.

Value

Nothing, the graph is draw on the current device.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Demsar, J. (2006) *Statistical Comparisons of Classifiers over Multiple Data Sets*. Journal of Machine Learning Research, 7, 1-30.

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

`CDdiagram.Nemenyi`, `CDdiagram.BD`, `signifDiffs`, `metricNames`, `performanceEstimation`, `topPerformers`, `topPerformer`, `rankWorkflows`, `metricsSummary`, `ComparisonResults`

Examples

```
## Not run:
## Estimating MSE for 3 variants of both
## regression trees and SVMs, on  two data sets, using one repetition
## of 10-fold CV
library(e1071)
data(iris)
data(Satellite,package="mlbench")
data(LetterRecognition,package="mlbench")


## running the estimation experiment
res <- performanceEstimation(
```

```
             c(PredTask(Species ~ .,iris),
               PredTask(classes ~ .,Satellite,"sat"),
               PredTask(lettr ~ .,LetterRecognition,"letter")),
             workflowVariants(learner="svm",
                   learner.pars=list(cost=1:4,gamma=c(0.1,0.01))),
             EstimationTask(metrics=c("err","acc"),method=CV()))


## checking the top performers
topPerformers(res)

## now let us assume that we will choose "svm.v2" as our baseline
## carry out the paired comparisons
pres <- pairedComparisons(res,"svm.v2")

## obtaining a CD diagram comparing all workflows against
## the baseline (defined in the previous call to pairedComparisons)
CDdiagram.BD(pres,metric="err")


## End(Not run)
```

`CDdiagram.Nemenyi` *CD diagrams for the post-hoc Nemenyi test*

Description

This function obtains a Critical Difference (CD) diagram for the post-hoc Nemenyi test in the lines defined by Demsar (2006). These diagrams provide an interesting visualization of the statistical significance of the observed paired differences between a set of workflows on a set of predictive tasks. They allow us to compare all workflows against each other on these set of tasks and check the results of all these paired comparisons.

Usage

```
CDdiagram.Nemenyi(r, metric = names(r)[1])
```

Arguments

`r`	A list resulting from a call to `pairedComparisons`
`metric`	The metric for which the CD diagram will be obtained (defaults to the first metric of the comparison).

Details

Critical Difference (CD) diagrams are interesting sucint visualizations of the results of a Nemenyi post-hoc test that is designed to check the statistical significance between the differences in average rank of a set of workflows on a set of predictive tasks.

In the resulting graph each workflow is represented by a colored line. The X axis where the lines end represents the average rank position of the respective workflow across all tasks. The null hypothesis is that the average ranks of each pair of workflows to not differ with statistical significance (at some confidence level defined in the call to pairedComparisons that creates the object used to obtain these graphs). Horizontal lines connect the lines of the workflows for which we cannot exclude the hypothesis that their average ranks is equal. Any pair of workflows whose lines are not connected with an horizontal line can be seen as having an average rank that is different with statistical significance. On top of the graph an horizontal line is shown with the required difference between the average ranks (known as the critical difference) for two pair of workflows to be considered significantly different.

Value

Nothing, the graph is draw on the current device.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Demsar, J. (2006) *Statistical Comparisons of Classifiers over Multiple Data Sets*. Journal of Machine Learning Research, 7, 1-30.

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

CDdiagram.Nemenyi, CDdiagram.BD, signifDiffs, performanceEstimation, metricNames, topPerformers, topPerformer, rankWorkflows, metricsSummary, ComparisonResults

Examples

```
## Not run:
## Estimating MSE for 3 variants of both
## regression trees and SVMs, on  two data sets, using one repetition
## of 10-fold CV
library(e1071)
data(iris)
data(Satellite,package="mlbench")
data(LetterRecognition,package="mlbench")


## running the estimation experiment
res <- performanceEstimation(
           c(PredTask(Species ~ .,iris),
             PredTask(classes ~ .,Satellite,"sat"),
             PredTask(lettr ~ .,LetterRecognition,"letter")),
           workflowVariants(learner="svm",
                 learner.pars=list(cost=1:4,gamma=c(0.1,0.01))),
           EstimationTask(metrics=c("err","acc"),method=CV()))
```

```
## checking the top performers
topPerformers(res)

## now let us assume that we will choose "svm.v2" as our baseline
## carry out the paired comparisons
pres <- pairedComparisons(res,"svm.v2")

## obtaining a CD diagram comparing all workflows against
## each other
CDdiagram.Nemenyi(pres,metric="err")


## End(Not run)
```

`classificationMetrics` *Calculate some standard classification evaluation metrics of predictive performance*

Description

This function is able to calculate a series of classification evaluation statistics given two vectors: one with the true target variable values, and the other with the predicted target variable values. Some of the metrics may require additional information to be given (see Details section).

Usage

```
classificationMetrics(trues,preds,
            metrics=NULL,
            benMtrx=NULL,
           allCls=unique(c(levels(as.factor(trues)),levels(as.factor(preds)))),
            posClass=allCls[1],
            beta=1)
```

Arguments

`trues`	A vector or factor with the true values of the target variable.
`preds`	A vector or factor with the predicted values of the target variable.
`metrics`	A vector with the names of the evaluation statistics to calculate (see Details section). If none is indicated (default) it will calculate all available metrics.
`benMtrx`	An optional cost/benefit matrix with numeric values representing the benefits (positive values) and costs (negative values) for all combinations of predicted and true values of the nominal target variable of the task. In this context, the matrix should have the dimensions C x C, where C is the number of possible class values of the classification task. Benefits (positive values) should be on the diagonal of the matrix (situations where the true and predicted values are equal,

i.e. the model predicted the correct class and thus should be rewarded for that), whilst costs (negative values) should be on all positions outside of the diagonal of the matrix (situations where the predicted value is different from the true class value and thus the model should incur on a cost for this wrong prediction). The function assumes the rows of the matrix are the predicted values while the columns are the true class values.

`allCls` An optional vector with the possible values of the nominal target variable, i.e. a vector with the classes of the problem. The default of this parameter is to infer these values from the given vector of true and predicted values values. However, if these are small vectors (e.g. you are evaluating your model on a small test set), it may happen that not all possible class values occur in this vector and this will potentially create problems in the sub-sequent calculations. Moreover, even if the vector is not small, for highly unbalanced classification tasks, this problem may still occur. In these contexts, it is safer to specifically indicate the possible class values through this parameter.

`posClass` An optional string with the name of the class (a value of the target nominal variable) that should be considered the "positive" class. This is used typically on two class problems where one of the classes is more relevant than the other (the positive class). It will default to the first value of the vector of possible classes (`allCls`).

`beta` An optional number for the value of the Beta parameter in the calulation of the F-measure (defaults to 1 that corresponds to giving equal relevance to precision and recall in the calculation of the F score).

Details

In the following description of the currently available metrics we denote the vector of true target variable values as t, the vector of predictions by p, while n denotes the size of these two vectors, i.e. the number of test cases. Furthermore we will denote the number of classes (different values of the target nominal variable) as C. For problems with only two classes, where one is considered the "positive" class we will use some extra notation, namely: TP = #p == + & t == + ; FP = #p == + & t == - ; TN = #p == - & t == -; FN = #p == - & t == + ; P = #t == + ; N = #t == -; Finally, for some descriptions we will use the concept of confusion matrix (CM). This is a C x C square matrix where entry CM[i,j] contains the number of times (for a certain test set) some model predicted class i for a true class value of j, i.e. rows of this matrix represent predicted values and columns true values. We will also refer to cost/benefit matrices (or utility matrices) that have the same structure (squared C x C) where entry CB[i,j] represents the cost/benefit of predicting a class i for a true class of j.

The currently available classification performance metrics are:

"acc": sum(I(t_i == p_i))/n, where I() is an indicator function given 1 if its argument is true and 0 otherwise. Note that "acc" is a value in the interval [0,1], 1 corresponding to all predictions being correct.

"err": the error rate, calculated as 1 - "acc"

"totU": this is a metric that takes into consideration not only the fact that the predictions are correct or not, but also the costs or benefits of these predictions. As mentioned above it assumes that the user provides a fully specified cost/benefit matrix though parameter `benMtrx`, with benefits corresponding to correct predictions, i.e. where t_i == p_i, while costs correspond to erroneous predictions. These matrices are C x C square matrices, where C is the number of possible values

of the nominal target variable (i.e. the number of classes). The entry benMtrx[x, y] represents the utility (a cost if x != y) of the model predicting x for a true value of y. The diagonal of these matrices corresponds to the correct predictions (t_i == p_i) and should have positive values (benefits). The positions outside of the diagonal correspond to prediction errors and should have negative values (costs). The "totU" measures the total Utility (sum of the costs and benefits) of the predictions of a classification model. It is calculated as sum(CB[p_i,t_j] * CM[p_i,t_j) where CB is a cost/benefit matrix and CM is a confusion matrix.

"fpr": false positives rate, is a metric applicable to two classes tasks that measures the proportion of times the model forecasted a positive class when it should not and it is given by FP/N

"fnr": false negatives rate, is a metric applicable to two classes tasks that measures the proportion of times the model forecasted a negative class when it should not, and it is given by FN/P

"tpr": true positives rate, is a metric applicable to two classes tasks that measures the proportion of times the model forecasted a positive class for the positive test cases, and it is given by TP/P

"tnr": true negatives rate, is a metric applicable to two classes tasks that measures the proportion of times the model forecasted a negative class for the negative test cases, and it is given by TN/N

"rec": recall, it is equal to the true positive rate ("tpr")

"sens": sensitivity, it is equal to the true positive rate ("tpr")

"spec": specificity, it is equal to the true negative rate ("tnr")

"prec": precision, it is a metric applicable to two classes tasks that measures the proportion of times the model forecasted a positive class and it was correct, and it is given by TP/(TP+FP)

"ppv": predicted positive value, it is equal to the precision ("prec")

"fdr": false discovery rate, it is a metric applicable to two classes tasks that is given by FP/(TP+FP)

"npv": negative predicted value, it is a metric applicable to two classes tasks that is given by TN/(TN+FN)

"for": false omission rate, it is a metric applicable to two classes tasks that is given by FN/(TN+FN)

"plr": positive likelihood ratio, it is a metric applicable to two classes tasks that is given by "tpr"/"fpr"

"nlr": negative likelihood ratio, it is a metric applicable to two classes tasks that is given by "fnr"/"tnr"

"dor": diagnostic odds ratio, it is a metric applicable to two classes tasks that is given by "plr"/"nlr"

"rpp": rate of positive predictions, it is a metric applicable to two classes tasks that measures the proportion of times the model forecasted a positive class, and it is given by (TP+FP)/N

"lift": lift, it is a metric applicable to two classes tasks and it is given by TP/P/(TP+FP) or equivalently TP/(P*TP+P*FP)

"F": the F-nmeasure, it is a metric applicable to two classes tasks that considers both the values of precision and recall weighed by a parameter Beta (defaults to 1 corresponding to equal weights to both), and it is given by (1+Beta^2)*("prec" * "rec") / ((Beta^2 * "prec") + "rec")

"microF": micro-averaged F-measure, it is equal to accuracy ("acc")

"macroF": macro-averaged F-measure, it is the average of the F-measure scores calculated by making the positive class each of the possible class values in turn

"macroRec": macro-averaged recall, it is the average recall by making the positive class each of the possible class values in turn

"macroPrec": macro-averaged precision, it is the average precision by making the positive class each of the possible class values in turn

Value

A named vector with the calculated statistics.

Note

1. In case you require "totU" to be calculated you must supply a cost/benefit matrix through parameter `benMtrx`.

2. If not all possible class values are present in the vector of true values in parameter `trues`, you should provide a vector with all the possible class values in parameter `allCls`.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

regressionMetrics

Examples

```
## Not run:
library(DMwR)
## Calculating several statistics of a classification tree on the Iris data
data(iris)
idx <- sample(1:nrow(iris),100)
train <- iris[idx,]
test <- iris[-idx,]
tree <- rpartXse(Species ~ .,train)
preds <- predict(tree,test,type='class')
## Calculate the  error rate
classificationMetrics(test$Species,preds)
## Calculate the all possible error metrics
classificationMetrics(test$Species,preds)
## Now trying calculating the utility of the predictions
cbM <- matrix(c(10,-20,-20,-20,20,-10,-20,-10,20),3,3)
classificationMetrics(test$Species,preds,"totU",cbM)

## End(Not run)
```

ComparisonResults-class *Class "ComparisonResults"*

Description

This is the main class that holds the results of performance estimation experiments involving several alternative workflows being applied and compared to several predictive tasks. For each workflow and task, a set of predictive performance metrics are estimated using some methodology and the results of this process are stored in these objets.

Objects from the Class

Objects can be created by calls of the form `ComparisonResults(...)`. These object are essentially a list of lists of objects of class `EstimationResults`. The top level is named list with has as many components as there are tasks. For each task there will be a named sub-list containing as many components as there are alternative workflows. Each of these components will contain and object of class `EstimationResults` with the estimation results for the particular workflow / task combination.

Methods

plot `signature(x = "ComparisonResults", y = "missing")`: plots the results of the experiments. It can result in an over-cluttered graph if too many workflows/tasks/evaluation metrics - use the subset method (see below) to overcome this.

show `signature(object = "ComparisonResults")`: shows the contents of an object in a proper way

subset `signature(x = "ComparisonResults")`: can be used to obtain a smaller ComparisonResults object containing only a subset of the information of the provided object. This method also accepts the arguments "tasks", "workflows" and "metrics". All are vectors of numbers or names that can be used to subset the original object. They default to all values of each dimension. See "methods?subset" for further details.

summary `signature(object = "ComparisonResults")`: provides a summary of the performance estimation experiment.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

`performanceEstimation`, `pairedComparisons`, `rankWorkflows`, `topPerformers`, `metricsSummary`, `mergeEstimationRes`

Examples

```
showClass("ComparisonResults")
## Not run:
## Estimating MAE, MSE, RMSE and MAPE for 3 variants of both
## regression trees and SVMs, on  two data sets, using one repetition
## of 10-fold CV
library(e1071)
library(DMwR)
data(swiss)
data(mtcars)

## running the estimation experiment
res <- performanceEstimation(
  c(PredTask(Infant.Mortality ~ .,swiss),PredTask(mpg ~ ., mtcars)),
  c(workflowVariants(learner="svm",
                     learner.pars=list(cost=c(1,10),gamma=c(0.01,0.5))),
    workflowVariants(learner="rpartXse",
                     learner.pars=list(se=c(0,0.5,1)))
  ),
  EstimationTask(metrics=c("mae","mse","rmse","mape"),method=CV())
  )

## Check a summary of the results
summary(res)

topPerformers(res)

summary(subset(res,metrics="mse"))
summary(subset(res,metrics="mse",partial=FALSE))
summary(subset(res,workflows="v1"))

## End(Not run)
```

`CV-class` *Class "CV"*

Description

This class of objects contains the information describing a cross validation experiment, i.e. its settings.

Objects from the Class

Objects can be created by calls of the form `CV(...)` providing the values for the class slots. These objects include information on the number of repetitions of the cross validation (CV) experiment, the number of folds, the random number generator seed, whether the sampling should or not be stratified and *optionally*, the concrete data splits to use on each repetition and iteration of the CV experiment. Note that most of the times you will not supply these data splits as the CV routines in

this infra-structure will take care of building them. Still, this allows you to replicate some experiment carried out with specific train/test splits.

Slots

`nReps`: Object of class `numeric` indicating the number of repetitions of the N folds CV experiment (defaulting to 1).

`nFolds`: Object of class `numeric` with the number of folds on each CV experiment (defaulting to 10).

`strat`: Object of class `logical` indicating whether the sampling should or not be stratified (defaulting to FALSE).

`seed`: Object of class `numeric` with the random number generator seed (defaulting to 1234).

`dataSplits`: Object of class `list` containing the data splits to use on each repetition of a k-folds CV experiment (defaulting to `NULL`). This list should contain `nReps` x `nFolds` elements. Each element should be a vector with the row ids of the test set of the respective iteration. For instance, on a 3 x 10-fold CV experiment the 10th element should contain the ids of the test cases of the 10th fold of the first repetition and the 11th element the ids of the test cases on the 1st fold of the 2nd repetition. On all these iterations the training set will be formed by the ids not appearing in the test set.

Extends

Class `EstCommon`, directly. Class `EstimationMethod`, directly.

Methods

show `signature(object = "CV")`: method used to show the contents of a `CV` object.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

`MonteCarlo`, `LOOCV`, `Bootstrap`, `Holdout`, `EstimationMethod`, `EstimationTask`

Examples

```
showClass("CV")

## the defaults (1 x 10-fold CV)
s <- CV()

## stratified 2 x 5-fold CV
```

```
s1 <- CV(nReps=2,nFolds=5,strat=TRUE)

## Small example illustrating the format of user supplied data splits.
## This could be a 3-fold CV process of a data set with 30 cases
s2 <- CV(dataSplits=list(1:10,11:20,21:30))
s2
```

`cvEstimates` *Performance estimation using cross validation*

Description

This function obtains cross validation estimates of performance metrics for a given predictive task and method to solve it (i.e. a workflow). The function is general in the sense that the workflow function that the user provides as the solution to the task can implement or call whatever modeling technique the user wants.

The function implements N x K-fold cross validation (CV) estimation. Different settings concerning this methodology are available through the argument `estTask` (check the help page of `EstimationTask` and `CV`).

Please note that most of the times you will not call this function directly (though there is nothing wrong in doing it) but instead you will use the function `performanceEstimation`, that allows you to carry out performance estimation for multiple workflows on multiple tasks, using the estimation method you want (e.g. cross validation). Still, when you simply want to have the CV estimate of one workflow on one task, you may prefer to use this function directly.

Usage

```
cvEstimates(wf,task,estTask,cluster)
```

Arguments

`wf`	an object of the class `Workflow` representing the modeling approach to be evaluated on a certain task.
`task`	an object of the class `PredTask` defining the prediction task for which we want estimates.
`estTask`	an object of the class `EstimationTask` indicating the metrics to be estimated and the cross validation settings to use.
`cluster`	an optional parameter that can either be `TRUE` or a `cluster`. In case of `TRUE` the function will run in parallel and will internally setup the parallel back-end (defaulting to using half of the cores in your local machine). You may also setup outside your parallel back-end (c.f. `makeCluster`) and then pass the resulting `cluster` object to this function using this parameter. In case no value is provided for this parameter the function will run sequentially.

Details

The idea of this function is to carry out a cross validation experiment with the goal of obtaining reliable estimates of the predictive performance of a certain approach to a predictive task. This approach (denoted here as a *workflow*) will be evaluated on the given predictive task using some user-selected metrics, and this function will provide k-fold cross validation estimates of the true values of these evaluation metrics. k-Fold cross validation estimates are obtained by randomly partitioning the given data set into k equal size sub-sets. Then a learn+test process is repeated k times. At each iteration one of the k partitions is left aside as test set and the model is obtained with a training set formed by the remaining k-1 partitions. The process is repeated leaving each time one of the k partitions aside as test set. In the end the average of the k scores obtained on each iteration is the cross validation estimate.

Parallel execution of the estimation experiment is only recommended for minimally large data sets otherwise you may actually increase the computation time due to communication costs between the processes.

Value

The result of the function is an object of class `EstimationResults`.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

`CV`, `Workflow`, `standardWF`, `PredTask`, `EstimationTask`, `performanceEstimation`, `hldEstimates`, `bootEstimates`, `loocvEstimates`, `mcEstimates`, `EstimationResults`

Examples

```
## Not run:

## Estimating the mean squared error  of svm on the swiss data,
## using two repetitions of 10-fold CV
library(e1071)
data(swiss)

## Now the evaluation
eval.res <- cvEstimates(
             Workflow(wf="standardWF", wfID="mySVMtrial",
                      learner="svm", learner.pars=list(cost=10,gamma=0.1)
                     ),
             PredTask(Infant.Mortality ~ ., swiss),
             EstimationTask(metrics="mse",method=CV(nReps=2,nFolds=10))
                       )
```

```
## Check a summary of the results
summary(eval.res)


## An example with a user-defined workflow function implementing a
## simple approach using linear regression models but also containing
## some data-preprocessing and well as results post-processing.
myLM <- function(form,train,test,k=10,.outModel=FALSE) {
    require(DMwR)
    ## fill-in NAs on both the train and test sets
    ntr <- knnImputation(train,k)
    nts <- knnImputation(test,k,distData=train)
    ## obtain a linear regression model and simplify it
    md <- lm(form,ntr)
    md <- step(md)
    ## get the model predictions
    p <- predict(md,nts)
    ## post-process the predictions (this is an example assuming the target
    ## variable is always positive so we change negative predictions into 0)
    p <- ifelse(p < 0,0,p)
    ## now get the final return object
    res <- list(trues=responseValues(form,nts), preds=p)
    if (.outModel) res <- c(res,list(model=m))
    res
}

## Now for the CV estimation
data(algae,package="DMwR")
eval.res2 <- cvEstimates(
             Workflow(wf="myLM",k=5),
             PredTask(a1 ~ ., algae[,1:12],"alga1"),
             EstimationTask("mse",method=CV()))

## Check a summary of the results
summary(eval.res2)

##
## Parallel execution example
##
## Comparing the time of sequential and parallel execution
## using half of the cores of the local machine
##
data(Satellite,package="mlbench")
library(e1071)
system.time({p <- cvEstimates(Workflow(learner="svm"),
                              PredTask(classes ~ .,Satellite),
                              EstimationTask("err",Boot(nReps=10)),
                              cluster=TRUE)})[3]
system.time({np <- cvEstimates(Workflow(learner="svm"),
                               PredTask(classes ~ .,Satellite),
                               EstimationTask("err",Boot(nReps=10)))})[3]
```

```
## End(Not run)
```

EstCommon-class *Class* "EstCommon"

Description

An auxiliar class defining slots common to all experimental settings classes.

Objects from the Class

Objects can be created by calls of the form `new("EstCommon", ...)`.

Slots

seed: Object of class "numeric"

dataSplits: Object of class "OptMatrix"

Methods

No methods defined with class "EstCommon" in the signature.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

CV, MonteCarlo, LOOCV, Bootstrap, Holdout, EstimationMethod, EstimationTask

Examples

```
showClass("EstCommon")
```

EstimationMethod-class

Class "EstimationMethod"

Description

This is a class union formed by the classes CvSettings, McSettings, HldSettings, LoocvSettings and BootSettings

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

No methods defined with class "EstimationMethod" in the signature.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

CV, MonteCarlo, LOOCV, Holdout, Bootstrap, EstimationTask

Examples

```
showClass("EstimationMethod")
```

EstimationResults-class

Class "EstimationResults"

Description

This is the class of the objects storing the results of estimating the performance of a workflow on a predictive task using some estimation method.

Objects from the Class

Objects can be created by calls of the form `EstimationResults(...)` providing the values for the class slots. The objects contain information on the predictive task, the workflow, the estimation task, the metric scores and optionally also information on results obtained at each iteration of the estimation process.

Slots

`task`: Object of class `PredTask`

`workflow`: Object of class `Workflow`

`estTask`: Object belonging to class `EstimationTask`

`iterationsScores`: Object of class `matrix`

`iterationsInfo`: Object of class `list`

Methods

plot `signature(x = "EstimationResults", y = "missing")`: method used to visualize the results of the estimation process.

show `signature(object = "EstimationResults")`: shows the contents of an object in a proper way

summary `signature(object = "EstimationResults")`: method used to obtain a summary of the results of the estimation process.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

`EstimationMethod`, `EstimationTask`, `PredTask`, `Workflow`, `performanceEstimation`

Examples

```
showClass("EstimationResults")

## Not run:
library(e1071)
data(swiss)

## Estimating the MAE and NMSE of an SVM on the swiss task
eval.res <- cvEstimates(
                Workflow(learner="svm",learner.pars=list(cost=10,gamma=0.1)),
                PredTask(Infant.Mortality ~ .,swiss),
```

```
                    EstimationTask(metrics=c("mae","nmse"),method=CV(nReps=2))
                          )

## Check a summary of the results
summary(eval.res)


## End(Not run)
```

`estimationSummary` *Obtain a set of descriptive statistics of the scores of a workflow on a task*

Description

This function provides a set of descriptive statistics for each evaluation metric that is estimated on a performance estimation comparison. These statistics are obtained for a particular workflow, and for one of the prediction tasks involved in the experiment.

Usage

```
estimationSummary(results,workflow,task)
```

Arguments

`results`	This is a `ComparisonResults` object (type "class?ComparisonResults" for details) that contains the results of a performance estimation experiment obtained through the `performanceEstimation()` function.
`workflow`	A string with the ID of a workflow (it can also be an integer).
`task`	A string with the ID of a task (it can also be an integer).

Value

The function returns a matrix with the rows representing summary statistics of the scores obtained by the model on the different iterations, and the columns representing the evaluation statistics estimated in the experiment.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

`getScores`, `performanceEstimation`

Examples

```
## Not run:
## Estimating MSE for 3 variants of both
## regression trees and SVMs, on  two data sets, using one repetition
## of 10-fold CV
library(e1071)
data(swiss)

## running the estimation experiment
res <- performanceEstimation(
  PredTask(Infant.Mortality ~ .,swiss),
  workflowVariants(learner="svm",
                   learner.pars=list(cost=c(1,10),gamma=c(0.01,0.5))),
  EstimationTask("mse",method=CV(nReps=2,nFolds=5))
  )

## Get the summary of the estimations of svm.v2 on swiss
estimationSummary(res,"svm.v2","swiss.Infant.Mortality")


## End(Not run)
```

EstimationTask-class *Class* "EstimationTask"

Description

This class of objects contains the information describing an estimation task.

Details

In case you are providing your own user-defined evaluator functions (through parameters `evaluator` anbd `evaluator.pars`) you need to follow some protocol in defining these functions so that the package may correctly call them during the execution of the estimation experiments. This protocol depends on the output of the workflows you plan to evaluate with your use-defined function. Standard workflows (`standardWF` or `timeseriesWF`) will return at least a vector named `trues` with the true values of the test cases and the vector named `preds` with the respective predicted values (in this order). This means that your evaluator function should assume that it will be called with these two vectors as the first two arguments. However, if you are not using the standard workflows, you have more flexibility. In effect, user-defined workflows return whatever the author wants them to return (unless they are going to be evaluated using either `classificationMetrics` or `regressionMetrics`, that require the vectors of true and predicted values). This means that in the most flexible case where you have your own user-defined workflow function and your user-defined evaluator function, you can use whatever parameters you want. The only thing you need to worry is to be aware that your user-defined evaluator function will be called with whatever your user-defined workflow has returned as result of its execution. Your user-defined evaluator function should calculate whatever metrics are indicated through the parameter `stats` that is a vector of strings. In case the slot `trainReq` is `TRUE` then the user-defined evaluator function should also have a parameter

named train.y that will "receive" the values of the target variable on the training set. The remaining parameters of the user-defined function can be freely defined by you and their values will be specified through the contents of the evaluator.pars list.

Objects from the Class

Objects can be created by calls of the form EstimationTask(...) providing the values for the class slots. These objects contain information on the metrics to be estimated, as well as on the estimation method to use to obtain the estimates. Moreover, in case you want to use metrics not currently implemented by this package you can also provide the name of a function (and its parameters) that will be called to calculate these metrics.

Slots

metrics: An optional vector of objects of class character containing the names of the metrics to be estimated. These can be any of the metrics provided by the functions classificationMetrics and regressionMetrics or "trTime", "tsTime" or "totTime" for training, testing and total time, respectively. You may also provide the name of any other metrics, but in that case you need to use the slots evaluator and evaluator.pars to indicate the function to be use to calculate them. If you do not supply the name of any metric, all metrics of the used evaluator function will be calculated.

method: Object of class EstimationMethod containing the estimation method and its settings to be used to obtain the estimates of the metrics (defaulting to CV()).

evaluator: An optional object of class character containing the name of a function to be used to calculate the specified metrics. It will default to either classificationMetrics or regressionMetrics depending on the type of prediction task.

evaluator.pars: An optional list containing the parameters to be passed to the function calculating the metrics.

trainReq: An optional logical value indicating whether the metrics to be calculated require that the training data is supplied (defaulting to FALSE). Note that if the user selects any of the metrics "nmse", "nmae" or "theil" this will be set to TRUE.

Methods

show signature(object = "EstimationTask"): method used to show the contents of a EstimationTask object.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

MonteCarlo, CV, LOOCV, Bootstrap, Holdout, EstimationMethod

Examples

```
showClass("EstimationTask")

## Estimating Mean Squared Error using 10-fold cross validation
et <- EstimationTask(metrics="mse")
et

## Estimating Accuracy and Total Time (train+test times) using 3
## repetitions of holdout with 20% of the cases used for testing.
EstimationTask(metrics=c("acc","totTime"),method=Holdout(nReps=3,hldSz=0.2))

## An example with a user-defined metric: the average differences between true
## predicted values raised to a certain power.

## First we define the function that calculates this metric. It
## needs to have 'trues' and 'preds' as the first two arguments if we
## want it to be usable by standard workflows
powErr <- function(trues,preds,metrics,pow=3) {
     if (metrics != "pow.err") stop("Unable to calculate that metric!")
     c(pow.err = mean((trues-preds)^pow))
}

## Now the estimation task (10-fold cv in this example)
EstimationTask(metrics="pow.err", method=CV(),
               evaluator="powErr", evaluator.pars=list(pow=4))
```

`getIterationsInfo` *Obtaining the information returned by a workflow when applied to a task, on a particular iteration of the estimation process or on all iterations*

Description

In the estimation process workflows are applied many times to different train+test samples of each task. We call these repetitions, the iterations of the estimation process. On each of these executions of the workflows, they typically return not only the predictions for the test set but also some extra information, in the form of a list. This function allows you to inspect this extra information

Usage

```
getIterationsInfo(obj, workflow = 1, task = 1, rep, fold, it)
```

Arguments

`obj`	A `ComparisonResults` object
`workflow`	A string with the ID of a workflow (it can also be an integer). It defaults to 1 (the first workflow of the estimation experiment)
`task`	A string with the ID of a task (it can also be an integer). It defaults to 1 (the first task of the estimation experiment)

rep	An integer representing the repetition, which allows you to identify the iteration you want to inspect. You need to specify either this argument together with the argument fold, or only the argument it
fold	An integer representing the fold, which allows you to identify the iteration you want to inspect. You need to specify either this argument together with the argument rep, or only the argument it
it	An integer representing the iteration you want to inspect. Alternatively, for cross validation experiments, you may instead specify the repetition id and the fold id (arguments rep and fold, respectivily)

Value

A list with the information returned by the workflow on the selected iteration or a list containing as many components as there are iterations (i.e. a list of lists) in case you have opted for obtaining all iterations results

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

getIterationsPreds, getScores, performanceEstimation

Examples

```
## Not run:
## Estimating MSE for 3 variants of both
## regression trees and SVMs, on  two data sets, using one repetition
## of 10-fold CV
library(e1071)
data(swiss)

## running the estimation experiment
res <- performanceEstimation(
  PredTask(Infant.Mortality ~ .,swiss),
  workflowVariants(learner="svm",
                   learner.pars=list(cost=c(1,10),gamma=c(0.01,0.5))),
  EstimationTask("mse",method=CV(nReps=2,nFolds=5))
  )

## Get the iterations scores of svm.v2 on swiss
getIterationsInfo(res,"svm.v2","swiss.Infant.Mortality",rep=1,fold=2)
## this would get the same
getIterationsInfo(res,"svm.v2","swiss.Infant.Mortality",it=2)
```

```
getIterationsInfo(res,"svm.v2","swiss.Infant.Mortality",rep=2,fold=3)
## this would get the same
getIterationsInfo(res,"svm.v2","swiss.Infant.Mortality",it=8)

## Get the results of all iterations
getIterationsInfo(res,"svm.v1","swiss.Infant.Mortality")


## End(Not run)
```

`getIterationsPreds` *Obtaining the predictions returned by a workflow when applied to a task, on a particular iteration of the estimation process, or on all iterations*

Description

In the estimation process workflows are applied many times to different train+test samples of each task. We call these repetitions, the iterations of the estimation process. On each of these executions of the workflows they must return the predictions for the test set. This function allows you to obtain these predictions. The function also allows you to obtain the predictions on all iterations, instead of a single iteration.

Usage

```
getIterationsPreds(obj, workflow = 1, task = 1, rep, fold, it, predComp="preds")
```

Arguments

`obj`	A `ComparisonResults` object
`workflow`	A string with the ID of a workflow (it can also be an integer). It defaults to 1 (the first workflow of the estimation experiment)
`task`	A string with the ID of a task (it can also be an integer). It defaults to 1 (the first task of the estimation experiment)
`rep`	An integer representing the repetition, which allows you to identify the iteration you want to inspect. You need to specify either this argument together with the argument `fold`, or only the argument `it`
`fold`	An integer representing the fold, which allows you to identify the iteration you want to inspect. You need to specify either this argument together with the argument `rep`, or only the argument `it`
`it`	An integer representing the iteration you want to inspect. Alternatively, for cross validation experiments, you may instead specify the repetition id and the fold id (arguments `rep` and `fold`, respectivily)
`predComp`	A string with the name of the component of the list returned by the workflow that contains the predictions (it defaults to "preds")

Value

The result is either a vector of the predictions of a particular iteration or a matrix with the predictions on all iterations

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

getScores, getIterationsInfo, performanceEstimation

Examples

```
## Not run:
## Estimating MSE for 3 variants of both
## regression trees and SVMs, on  two data sets, using one repetition
## of 10-fold CV
library(e1071)
data(swiss)

## running the estimation experiment
res <- performanceEstimation(
  PredTask(Infant.Mortality ~ .,swiss),
  workflowVariants(learner="svm",
                   learner.pars=list(cost=c(1,10),gamma=c(0.01,0.5))),
  EstimationTask("mse",method=CV(nReps=2,nFolds=5))
  )

## Get the iterations scores of svm.v2 on swiss
getIterationsPreds(res,"svm.v2","swiss.Infant.Mortality",rep=1,fold=2)
## this would get the same
getIterationsPreds(res,"svm.v2","swiss.Infant.Mortality",it=2)

getIterationsPreds(res,"svm.v2","swiss.Infant.Mortality",rep=2,fold=3)
## this would get the same
getIterationsPreds(res,"svm.v2","swiss.Infant.Mortality",it=8)

## Get the results of all iterations
getIterationsPreds(res,"svm.v1","swiss.Infant.Mortality")


## End(Not run)
```

getScores *Obtaining the metric scores on the different iterations for a workflow / task combination*

Description

With this function we can obtain the different metric scores obtained by a given workflow on a given task, in the different iterations of a performance estimation experiment.

Usage

```
getScores(results, workflow, task)
```

Arguments

results	A ComparisonResults object
workflow	A string with the ID of a workflow
task	A string with the ID of a predictive task

Value

A matrix with as many rows as there are iterations and as many columns as there are metrics being estimated in the experiment

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

taskNames,workflowNames,metricNames,estimationSummary

Examples

```
## Not run:
## Estimating MSE for 3 variants of both
## regression trees and SVMs, on  two data sets, using one repetition
## of 10-fold CV
library(e1071)
library(DMwR)
data(swiss)
data(mtcars)
```

```
## running the estimation experiment
res <- performanceEstimation(
  c(PredTask(Infant.Mortality ~ .,swiss),PredTask(mpg ~ ., mtcars)),
  c(workflowVariants(learner="svm",
                     learner.pars=list(cost=c(1,10),gamma=c(0.01,0.5))),
    workflowVariants(learner="rpartXse",
                     learner.pars=list(se=c(0,0.5,1)))
  ),
  EstimationTask("mse")
  )

## Get the iterations scores of svm.v2 on swiss
getScores(res,"svm.v2","swiss.Infant.Mortality")


## End(Not run)
```

`getWorkflow` *Obtain the workflow object corresponding to an ID*

Description

This function can be used to obtain the `Workflow` object corresponding to an ID used in a performance estimation experiment. This allows you for instance to check the full details of the workflow corresponding to that ID (e.g. the function implementing the workflow, the parameters and their values, etc.)

Usage

```
getWorkflow(var, obj)
```

Arguments

`var`	The string with the workflow ID
`obj`	A `ComparisonResults` object with the data from a performance estimation experiment

Value

A `Workflow` object

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

Workflow, runWorkflow, performanceEstimation

Examples

```
## Not run:
## Estimating MSE for 3 variants of both
## regression trees and SVMs, on  two data sets, using one repetition
## of 10-fold CV
library(e1071)
library(DMwR)
data(swiss)
data(mtcars)

## running the estimation experiment
res <- performanceEstimation(
  c(PredTask(Infant.Mortality ~ .,swiss),PredTask(mpg ~ ., mtcars)),
  c(workflowVariants(learner="svm",
                     learner.pars=list(cost=c(1,10),gamma=c(0.01,0.5))),
    workflowVariants(learner="rpartXse",
                     learner.pars=list(se=c(0,0.5,1)))
  ),
  EstimationTask("mse",method=CV(nReps=2,nFolds=5))
  )

## Get the workflow corresponding to the ID svm.v2
getWorkflow("svm.v2",res)

## End(Not run)
```

---

hldEstimates *Performance estimation using holdout and random resampling*

Description

This function obtains hold-out and random sub-sampling estimates of performance metrics for a given predictive task and method to solve it (i.e. a workflow). The function is general in the sense that the workflow function that the user provides as the solution to the task, can implement or call whatever modeling technique the user wants.

The function implements hold-out and random sub-sampling (repeated hold-out) estimation. Different settings concerning this methodology are available through the argument `estTask` (check the help page of Holdout).

Please note that most of the times you will not call this function directly (though there is nothing wrong in doing it) but instead you will use the function performanceEstimation, that allows you to carry out performance estimation for multiple workflows on multiple tasks, using some estimation method you want (e.g. hold-out). Still, when you simply want to have the hold-out estimate of one workflow on one task, you may prefer to use this function directly.

Usage

```
hldEstimates(wf,task,estTask,cluster)
```

Arguments

wf	an object of the class Workflow representing the modeling approach to be evaluated on a certain task.
task	an object of the class PredTask representing the prediction task to be used in the evaluation.
estTask	an object of the class EstimationTask representing the hold-out settings to use.
cluster	an optional parameter that can either be TRUE or a cluster. In case of TRUE the function will run in parallel and will internally setup the parallel back-end (defaulting to using half of the cores in your local machine). You may also setup outside your parallel back-end (c.f. makeCluster) and then pass the resulting cluster object to this function using this parameter. In case no value is provided for this parameter the function will run sequentially.

Details

The idea of this function is to carry out a hold-out experiment with the goal of obtaining reliable estimates of the predictive performance of a certain approach to a predictive task. This approach (denoted here as a *workflow*) will be evaluated on the given predictive task using some user-selected metrics, and this function will provide hold-out or random sub-sampling estimates of the true value of these evaluation metrics. Hold-out estimates are obtained by randomly partition the given data set into train and test sets. The training set is used to obtain a model for the predictive task, which is then tested by making predictions for the test set. This random split of the given data can be repeated several times leading to what is usually known as random sub-sampling estimates. In the end the average of the scores over the several repetitions (if using *pure* hold-out this is only one) are the hold-out estimates of the selected metrics.

Parallel execution of the estimation experiment is only recommended for minimally large data sets otherwise you may actually increase the computation time due to communication costs between the processes.

Value

The result of the function is an object of class EstimationResults.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

Holdout, Workflow, standardWF, PredTask, EstimationTask, performanceEstimation, cvEstimates, bootEstimates, loocvEstimates, mcEstimates, EstimationResults

Examples

```
## Not run:

## Estimating the mean absolute error and the normalized mean squared
## error of rpart on the swiss data, using 70%-30% hold-out
library(e1071)
data(swiss)

## Now the evaluation
eval.res <- hldEstimates(
             Workflow(wf="standardWF",wfID="svmApproach",
                      learner="svm",learner.pars=list(cost=10,gamma=0.1)
                     ),
             PredTask(Infant.Mortality ~ ., swiss),
             EstimationTask(metrics=c("mae","nmse"),
                            method=Holdout(nReps=5,hldSz=0.3))
                       )

## Check a summary of the results
summary(eval.res)

## An example with a user-defined workflow function implementing a
## simple approach using linear regression models but also containing
## some data-preprocessing and well as results post-processing.
myLM <- function(form,train,test,k=10,.outModel=FALSE) {
    require(DMwR)
    ## fill-in NAs on both the train and test sets
    ntr <- knnImputation(train,k)
    nts <- knnImputation(test,k,distData=train)
    ## obtain a linear regression model and simplify it
    md <- lm(form,ntr)
    md <- step(md)
    ## get the model predictions
    p <- predict(md,nts)
    ## post-process the predictions (this is an example assuming the target
    ## variable is always positive so we change negative predictions into 0)
    p <- ifelse(p < 0,0,p)
    ## now get the final return object
    res <- list(trues=responseValues(form,nts), preds=p)
    if (.outModel) res <- c(res,list(model=m))
    res
}

## Now for the Holdout estimation
data(algae,package="DMwR")
eval.res2 <- hldEstimates(
             Workflow(wf="myLM",k=5),
```

```
            PredTask(a1 ~ ., algae[,1:12],"alga1"),
            EstimationTask("mse",method=Holdout(nReps=5)))

## Check a summary of the results
summary(eval.res2)

## End(Not run)
```

`Holdout-class` *Class "Holdout"*

Description

This class of objects contains the information describing a hold out experiment, i.e. its settings.

Objects from the Class

Objects can be created by calls of the form `Holdout(...)` providing the values for the class slots. The objects contain information on the number of repetitions of the hold out experiment, the percentage of the given data to set as hold out test set, the random number generator seed, information on whether stratified sampling should be used and *optionally* the concrete data splits to use on each iteration of the holdout experiment. Note that most of the times you will not supply these data splits as the holdout routines in this infra-structure will take care of building them. Still, this allows you to replicate some experiment carried out with specific train/test splits.

Slots

`nReps`**:** Object of class `numeric` indicating the number of repetitions of the N folds CV experiment (defaulting to 1).

`hldSz`**:** Object of class `numeric` with the percentage (a number between 0 and 1) of cases to use as hold out (defaulting to 0.3).

`strat`**:** Object of class `logical` indicating whether the sampling should or not be stratefied (defaulting to FALSE).

`seed`**:** Object of class `numeric` with the random number generator seed (defaulting to 1234).

`dataSplits`**:** Object of class `list` containing the data splits to use on each repetition of a `nReps` Holdout experiment (defaulting to `NULL`). This list should contain `nReps` elements. Each element should be a vector with the row ids of the test set of the respective iteration. On all these iterations the training set will be formed by the ids not appearing in the test set.

Extends

Class `EstCommon`, directly. Class `EstimationMethod`, directly.

Methods

show `signature(object = "Holdout")`: method used to show the contents of a `Holdout` object.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

MonteCarlo, LOOCV, Bootstrap, CV, EstimationMethod, EstimationTask

Examples

```
showClass("Holdout")

## 10 repetitions of a holdout experiment leaving on each repetition
## 20% of the cases randomly chosen as test set (the holdout)
h1 <- Holdout(nReps=10,hldSz=0.2,strat=TRUE)
h1

## Small example illustrating the format of user supplied data splits
## in this case for 3 repetitions of a Holdout process where each test
## set has 10 cases
h2 <- Holdout(dataSplits=list(1:10,11:20,21:30))
h2
```

---

is.classification *Check if a certain predictive task is a classification problem*

Description

This function tests if a task defined by a formula over a data set is a classification task, which will be the case if the target variable is nominal.

Usage

```
is.classification(task)
```

Arguments

task	An object of class PredTask

Value

A logical value

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

is.regression

Examples

```
data(iris)
tsk <- PredTask(Species ~ .,iris)
if (is.classification(tsk)) cat("This is a classification task.\n")
```

---

is.regression *Check if a certain predictive task is a regression problem*

Description

This function tests if a task defined by a formula over a data set is a regression task, which will be the case if the target variable is numeric.

Usage

```
is.regression(task)
```

Arguments

task	An object of class PredTask

Value

A logical value

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

is.classification

Examples

```
data(iris)
tsk <- PredTask(Species ~ .,iris)
if (!is.regression(tsk)) cat("This is not a regression task!\n")
```

---

knnImp *Fill in NA values with the values of the nearest neighbours*

Description

Function that fills in all NA values using the k Nearest Neighbours of each case with NA values. It uses the median/most frequent value within the neighbours to fill in the NAs.

Usage

```
knnImp(data, k = 10, scale = TRUE, distData = NULL)
```

Arguments

data	A data frame with the data set
k	The number of nearest neighbours to use (defaults to 10)
scale	Boolean setting if the data should be scale before finding the nearest neighbours (defaults to TRUE)
distData	Optionally you may sepecify here a data frame containing the data set that should be used to find the neighbours. This is usefull when filling in NA values on a test set, where you should use only information from the training set. This defaults to NULL, which means that the neighbours will be searched in data

Details

This function uses the k-nearest neighbours to fill in the unknown (NA) values in a data set. For each case with any NA value it will search for its k most similar cases and use the values of these cases to fill in the unknowns.

The function will use either the median (in case of numeric variables) or the most frequent value (in case of factors), of the neighbours to fill in the NAs.

Value

A data frame without NA values

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

na.omit

Examples

```
## Not run:
data(algae,package="DMwR")
cleanAlgae <- knnImp(algae)
summary(cleanAlgae)

## End(Not run)
```

LOOCV-class *Class "LOOCV"*

Description

This class of objects contains the information describing a leave one out cross validation estimation experiment, i.e. its settings.

Objects from the Class

Objects can be created by calls of the form `LOOCV(...)` providing the values for the class slots. These objects contain infoprmation on the random number generator seed and also whether the execution of the experiments should be verbose.

Slots

`seed`**:** Object of class `numeric` with the random number generator seed (defaulting to 1234).

`dataSplits`**:** Object of class `list` containing the data splits to use on each repetition of a leave one out cross validation experiment (defaulting to `NULL`). This list should contain as many elements as there are cases in the task data set. Each element should be the row id of the test case of the respective iteration. On all these iterations the training set will be formed by the remaining ids.

Extends

Class EstCommon, directly. Class EstimationMethod, directly.

Methods

show `signature(object = "LOOCV")`: method used to show the contents of a `LOOCV` object.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

`MonteCarlo`, `CV`, `Bootstrap`, `Holdout`, `EstimationMethod`, `EstimationTask`

Examples

```
showClass("LOOCV")

s <- LOOCV()
s
```

---

loocvEstimates *Performance estimation using Leave One Out Cross Validation*

Description

This function obtains leave one out cross validation estimates of performance metrics for a given predictive task and method to solve it (i.e. a workflow). The function is general in the sense that the workflow function that the user provides as the solution to the task, can implement or call whatever modeling technique the user wants.

The function implements leave one out cross validation estimation. Different settings concering this methodology are available through the argument `estTask` (check the help page of `LOOCV`).

Please note that most of the times you will not call this function directly, though there is nothing wrong in doing it, but instead you will use the function `performanceEstimation`, that allows you to carry out performance estimation of multiple workflows on multiple tasks, using some estimation method like for instance cross validation. Still, when you simply want to have the leave one out cross validation estimate of one workflow on one task, you may use this function directly.

Usage

```
loocvEstimates(wf,task,estTask,verbose=FALSE,cluster)
```

Arguments

wf	an object of the class Workflow representing the modeling approach to be evaluated on a certain task.
task	an object of the class PredTask representing the prediction task to be used in the evaluation.
estTask	an object of the class EstimationTask indicating the metrics to be estimated and the leave one out cross validation settings to use.
verbose	A boolean value controlling the level of output of the function execution, defaulting to FALSE
cluster	an optional parameter that can either be TRUE or a cluster. In case of TRUE the function will run in parallel and will internally setup the parallel back-end (defaulting to using half of the cores in your local machine). You may also setup outside your parallel back-end (c.f. makeCluster) and then pass the resulting cluster object to this function using this parameter. In case no value is provided for this parameter the function will run sequentially.

Details

The idea of this function is to carry out a leave one out cross validation experiment with the goal of obtaining reliable estimates of the predictive performance of a certain approach to a predictive task. This approach (denoted here as a *workflow*) will be evaluated on the given predictive task using some user-selected metrics, and this function will provide leave one out cross validation estimates of the true value of these evaluation metrics. Leave one out cross validation estimates are obtained as the average of *N* iterations, where *N* is the size of the given data sample. On each of these iterations one of the cases in the data sample is left out as *test set* and the worflow is applied to the remaining *N-1* cases. The process is repeated for all cases, i.e. *N* times. This estimation is similar to k-fold cross validation where k equals to *N*. The resulting estimates are obtained by averaging over the *N* iteration scores.

Parallel execution of the estimation experiment is only recommended for minimally large data sets otherwise you may actually increase the computation time due to communication costs between the processes.

Value

The result of the function is an object of class EstimationResults.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

LOOCV, Workflow, standardWF, PredTask, EstimationTask, performanceEstimation, hldEstimates, bootEstimates, cvEstimates, mcEstimates, EstimationResults

Examples

```
## Not run:

## Estimating the error rate of an SVM on the iris data set using
## leave one out cross validation
library(e1071)
data(iris)

## Now the evaluation
eval.res <- loocvEstimates(
             Workflow(wfID="svmTrial",
                      learner="svm",learner.pars=list(cost=10,gamma=0.1)
                     ),
             PredTask(Species ~ ., iris),
             EstimationTask("err",method=LOOCV()))

## Check a summary of the results
summary(eval.res)


## End(Not run)
```

`mcEstimates` *Performance estimation for time series prediction tasks using Monte Carlo*

Description

This function performs a Monte Carlo experiment with the goal of estimating the performance of a given approach (a workflow) on a certain time series prediction task. The function is general in the sense that the workflow function that the user provides as the solution to the task, can implement or call whatever modeling technique the user wants.

The function implements Monte Carlo estimation and different settings concerning this methodology are available through the argument `estTask` (check the help page of MonteCarlo).

Please note that most of the times you will not call this function directly, though there is nothing wrong in doing it, but instead you will use the function performanceEstimation, that allows you to carry out performance estimation for multiple workflows on multiple tasks, using some estimation method. Still, when you simply want to have the Monte Carlo estimates for one workflow on one task, you may prefer to use this function directly.

Usage

```
mcEstimates(wf, task, estTask, verbose = TRUE,cluster)
```

Arguments

wf	an object of the class Workflow representing the modeling approach to be evaluated on a certain task.
task	an object of the class PredTask representing the prediction task to be used in the evaluation.
estTask	an object of the class EstimationTask indicating the metrics to be estimated and the Monte Carlo settings to use.
verbose	A boolean value controlling the level of output of the function execution, defaulting to TRUE
cluster	an optional parameter that can either be TRUE or a cluster. In case of TRUE the function will run in parallel and will internally setup the parallel back-end (defaulting to using half of the cores in your local machine). You may also setup outside your parallel back-end (c.f. makeCluster) and then pass the resulting cluster object to this function using this parameter. In case no value is provided for this parameter the function will run sequentially.

Details

This function provides reliable estimates of a set of evaluation statistics through a Monte Carlo experiment. The user supplies a worflow function and a data set of a time series forecasting task, together with the estimation task. This task should include both the metrics to be estimated as well as the settings of the estimation methodology (MOnte Carlo) that include, among others, the size of the training (TR) and testing sets (TS) and the number of repetitions (R) of the train+test cycle. The function randomly selects a set of R numbers in the time interval [TR+1,NDS-TS+1], where NDS is the size of the full data set. For each of these R numbers the previous TR observations of the data set are used to learn a model and the subsequent TS observations for testing it and obtaining the wanted evaluation metrics. The resulting R estimates of the evaluation metrics are averaged at the end of this process resulting in the Monte Carlo estimates of these metrics.

This function is targeted at obtaining estimates of performance for time series prediction problems. The reason is that the experimental repetitions ensure that the order of the rows in the original data set are never swaped, as these rows are assumed to be ordered by time. This is an important issue to ensure that a prediction model is never tested on past observations of the time series.

For each train+test iteration the provided workflow function is called and should return the predictions of the workflow for the given test period. To carry out this train+test iteration the user may use the standard time series workflow that is provided (check the help page of timeseriesWF), or may provide hers/his own workflow that should return a list as result. See the Examples section below for an example of these functions. Further examples are given in the package vignette.

Parallel execution of the estimation experiment is only recommended for minimally large data sets otherwise you may actually increase the computation time due to communication costs between the processes.

Value

The result of the function is an object of class EstimationResults.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

MonteCarlo, Workflow, timeseriesWF, PredTask, EstimationTask, performanceEstimation, hldEstimates, loocvEstimates, cvEstimates, bootEstimates, EstimationResults

Examples

```
## The following is a small illustrative example using the quotes of the
## SP500 index. This example estimates the performance of a random
## forest on a illustrative example of trying to forecast the future
## variations of the adijusted close prices of the SP500 using a few
## predictors. The random forest is evaluated on 4 repetitions of a
## monte carlo experiment where 30% of the data is used for training
## the model that is then used to make predictions for the next 20%,
## using a sliding window approach with a relearn step of 10 periods
## (check the help page of the timeseriesWF() function to understand
## these and other settings)

## Not run:
library(quantmod)
library(randomForest)

getSymbols('^GSPC',from='2008-01-01',to='2012-12-31')
data.model <- specifyModel(Next(100*Delt(Ad(GSPC))) ~ Delt(Ad(GSPC),k=1:10)+Delt(Vo(GSPC),k=1:3))
data <- as.data.frame(modelData(data.model))
colnames(data)[1] <- 'PercVarClose'

spExp <- mcEstimates(Workflow("timeseriesWF",wfID="rfTrial",
                              type="slide",relearn.step=10,
                              learner='randomForest'),
                     PredTask(PercVarClose ~ .,data,"sp500"),
                     EstimationTask(metrics=c("mse","theil"),
                                    method=MonteCarlo(nReps=4,szTrain=.3,szTest=.2)))

summary(spExp)

## End(Not run)
```

mergeEstimationRes *Merging several* ComparisonResults *class objects*

Description

This function can be used to join several objects of class ComparisonResults into a single object. The merge is carried out assuming that there is something in common between the objects (e.g. all use the same workflows on different tasks), and that the user specifies which property should be used for the merging process.

Usage

```
mergeEstimationRes(..., by = "tasks")
```

Arguments

...	The ComparisonResults class object names separated by commas
by	The dimension of the ComparisonResults class objects that should be used for the merge. All objects should have the same values on the remaining dimensions of an estimation experiment. For instance, if you merge by "tasks" (the default) it means that the objects being merged should include estimation results on the same set of workflows on the same set of metrics, using the same estimation method and settings. The only thing that changes between the objects in this example is the set of tasks. Possible values of this argument are: "tasks", "workflows" and "metrics".

Details

The objects of class ComparisonResults (type "class?ComparisonResults" for details) contain several information on the results of an estimation expriment for several workflows on several predictive tasks. Sometimes, when you are trying too many workflows on too many tasks, it is convinient to run these variants on different calls to the function performanceEstimation. After all calls are completed we frequently want to have all results on a single object. This is the objective of the current function: allow you to merge these different ComparisonResults objects into a single one. For being mergeable the objects need to have things in common otherwise it makes no sense to merge them. For instance, we could split our very large experiment by calling performanceEstimation with different tasks, although the rest (the workflows and the estimation task) stays the same. See the Examples section for some illustrations.

Value

The result of this function is a ComparisonResults object.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

performanceEstimation, ComparisonResults, subset

Examples

```
## Not run:
## Run some experiments with the swiss data and two different
## prediction models
data(swiss)

exp1 <- performanceEstimation(
  PredTask(Infant.Mortality ~ .,swiss),
  workflowVariants(learner="svm",
                   learner.pars=list(cost=c(1,10),gamma=c(0.01,0.5))),
  EstimationTask("mse")
  )

exp2 <- performanceEstimation(
  PredTask(Infant.Mortality ~ .,swiss),
  Workflow(learner="lm"),
  EstimationTask("mse")
  )


## joining the two experiments by workflows
all <- mergeEstimationRes(exp1,exp2,by="workflows")
topPerformers(all) # check the best results

## now an example by adding new metrics
exp3 <- performanceEstimation(
  PredTask(Infant.Mortality ~ .,swiss),
  Workflow(learner="lm"),
  EstimationTask(metrics=c("mae","totTime"))
  )

allLM <- mergeEstimationRes(exp2,exp3,by="metrics")
topPerformers(allLM)


## End(Not run)
```

---

metricNames *The evaluation metrics estimated in an experiment*

Description

This function can be used to get a vector with the names of the evaluation metrics that were used in a performance estimation experiment.

Usage

```
metricNames(o)
```

Arguments

o	An object of class ComparisonResults

Value

A vector of strings (the names of the metrics)

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

ComparisonResults, performanceEstimation, taskNames, workflowNames

Examples

```
## Not run:
## Estimating MSE for 3 variants of both
## regression trees and SVMs, on  two data sets, using one repetition
## of 10-fold CV
library(e1071)
library(DMwR)
data(swiss)
data(mtcars)

## running the estimation experiment
res <- performanceEstimation(
  c(PredTask(Infant.Mortality ~ .,swiss),PredTask(mpg ~ ., mtcars)),
  c(workflowVariants(learner="svm",
                     learner.pars=list(cost=c(1,10),gamma=c(0.01,0.5))),
    workflowVariants(learner="rpartXse",
                     learner.pars=list(se=c(0,0.5,1)))
  ),
  EstimationTask("mse")
  )
```

```
## the names of the metrics that were estimated in the above experiment
metricNames(res)

## End(Not run)
```

`metricsSummary` *Obtains a summary of the individual metric scores obtained by each workflow on a set of tasks.*

Description

Given a `ComparisonResults` object this function provides a summary statistic (defaulting to the mean) of the individual scores obtained on a each evaluation metric over all repetitions carried out in the estimation process. This is done for all workflows and tasks of the performance estimation experiment. The function can be handy to obtain things like for instance the maximum score obtained by each workflow on a particular metric over all repetitions of the experimental process. It is also usefull (using its defaults) as a way to quickly getting the estimated values for each metric obtained by each alternative workflow and task (see the Examples section).

Usage

```
metricsSummary(compRes, summary = "mean", ...)
```

Arguments

`compRes`	An object of class `ComparisonResults` with the results of a performance estimation experiment.
`summary`	A string with the name of the function that you want to use to obtain the summary (defaults to "mean"). This function will be applied to the set of individual scores of each workflow on each task and for all metrics.
`...`	Further arguments passed to the selected summary function.

Value

The result of this function is a named list with as many components as there are predictive tasks. For each task (component), we get a matrix with as many columns as there are workflows and as many rows as there are evaluation metrics. The values on this matrix are the results of applying the selected summary function to the metric scores on each iteration of the estimation process.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

performanceEstimation, topPerformers, topPerformer, rankWorkflows

Examples

```
## Not run:
## Estimating several evaluation metrics on different variants of a
## regression tree and of a SVM, on  two data sets, using one repetition
## of  10-fold CV

data(swiss)
data(mtcars)
library(e1071)

## run the experimental comparison
results <- performanceEstimation(
               c(PredTask(Infant.Mortality ~ ., swiss),
                 PredTask(mpg ~ ., mtcars)),
               c(workflowVariants(learner='svm',
                                  learner.pars=list(cost=c(1,5),gamma=c(0.1,0.01))
                                 )
               ),
               EstimationTask(metrics=c("mse","mae"),method=CV(nReps=2,nFolds=5))
                                 )

## Get the minium value of each metric on all iterations of the CV
## process.
metricsSummary(results,summary="min")

## Get a summary table for each task with the estimated scores for each
## metric by each workflow
metricsSummary(results)

## End(Not run)
```

MonteCarlo-class *Class "MonteCarlo"*

Description

This class of objects contains the information describing a monte carlo experiment, i.e. its settings.

Objects from the Class

Objects can be created by calls of the form `MonteCarlo(...)` providing the values for the class slots. These objects contain information on the number of repetitions of the experiments, the data used for training the models on each repetition, the data used for testing these models, the random number generator seed and *optionally* the concrete data splits to use on each iteration of the Monte Carlo experiment. Note that most of the times you will not supply these data splits as the Monte

Carlo routines in this infra-structure will take care of building them. Still, this allows you to replicate some experiment carried out with specific train/test splits.

Slots

`nReps`: Object of class `numeric` indicating the number of repetitions of the Monte Carlo experiment (defaulting to 10).

`szTrain`: Object of class `numeric`. If it is a value between 0 and 1 it is interpreted as a percentage of the available data set, otherwise it is interpreted as the number of cases to use. It defaults to 0.25.

`szTest`: Object of class `numeric` If it is a value between 0 and 1 it is interpreted as a percentage of the available data set, otherwise it is interpreted as the number of cases to use. It defaults to 0.25.

`seed`: Object of class `numeric` with the random number generator seed (defaulting to 1234).

`dataSplits`: Object of class `list` containing the data splits to use on each Monte Carlo repetition. Each element should be a list with two components: `test` and `train`, on this order. Each of these is a vector with the row ids to use as test and train sets of each repetition of the Monte Carlo experiment.

Extends

Class `EstCommon`, directly. Class `EstimationMethod`, directly.

Methods

show `signature(object = "MonteCarlo")`: method used to show the contents of a `MonteCarlo` object.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

`LOOCV`, `CV`, `Bootstrap`, `Holdout`, `EstimationMethod`, `EstimationTask`

Examples

```
showClass("MonteCarlo")

m1 <- MonteCarlo(nReps=10,szTrain=0.3,szTest=0.2)
m1

## Small example illustrating the format of user supplied data splits
```

```
## it assumes that the source data is formed by 10 cases and that each
## model is trainined with 3 cases and tested in the following case.
## This is obviously a unrealistic example in terms of size but
## illustrates the format of the data splits
m2 <- MonteCarlo(dataSplits=list(list(test=sample(1:150,50),train=sample(1:150,50)),
                                 list(test=sample(1:150,50),train=sample(1:150,50)),
                                 list(test=sample(1:150,50),train=sample(1:150,50))
                                ))
m2
```

`pairedComparisons` *Statistical hypothesis testing on the observed paired differences in estimated performance.*

Description

This function analyses the statistical significance of the paired comparisons between the estimated performance scores of a set of workflows. When you run the `performanceEstimation()` function to compare a set of workflows over a set of problems you obtain estimates of their performances across these problems. This function implements several statistical tests that can be used to test several hypothesis concerning the observed differences in performance between the workflows on the tasks.

Usage

```
pairedComparisons(obj,baseline,
                  maxs=rep(FALSE,length(metricNames(obj))),
                  p.value=0.05)
```

Arguments

`obj`	An object of class `ComparisonResults` that contains the results of a performance estimation experiment.
`baseline`	Several tests involve the hypothesis that a certain workflow is significantly different from a set of other alternatives. This argument allows you to specify the name of this baseline workflow. If you omit this name the function will default to the name of the workflow that has the lower average rank position across all tasks, for each estimation metric.
`maxs`	A vector of booleans with as many elements are there are metrics estimated in the experiment. A `TRUE` value means the respective metric is to be maximized, while a `FALSE` means minimization. Defaults to all `FALSE` values, i.e. all metrics are to be minimized.
`p.value`	A p value to be used in the calculations that involve using values from statistical tables (defaults to 0.05).

Details

The performanceEstimation function allows you to obtain estimates of the expected value of a series of performance metrics for a set of alternative workflows and a set of predictive tasks. After running this type of experiments we frequently want to check if there is any statistical significance between the estimated performance of the different workflows. The current function allows you to carry out this type of checks.

The function will only run on experiments containing more than one workflow as paired comparisons do not make sense with one single alternative workflow. Having more than one workflow we can distinguish two situations: i) comparing the performance of two workflows; or ii) comparisons among multiple workflows. The recommendations for checking the statistical significance of the difference between the performance of the alternative workflows varies within these two setups (see Demsar (2006) for recommendations).

The current function implements several statistical tests that can be used for different hypothesis tests. Namely, it obtains the results of paired *t* tests and paired *Wilcoxon Signed Rank* tests for situations where you are comparing the performance of two workflows, with the latter being recommended given the typical overlap among the training sets that does not ensure independence among the scores of the different iterations. For the setup of multiple workflows on multiple tasks the function also calculates the *Friedman* test and the post-hoc *Nemenyi* and *Bonferroni-Dunn* tests, according to the procedures described in Demsar (2006). The combination *Friedman* test followed by the post-hoc *Nemenyi* test is recommended when you want to carry out paired comparisons between all alternative workflows on the set of tasks to check for which differences are significant. The combination *Friedman* test followed by the post-hoc *Bonferroni-Dunn* test is recommended when you want to compare a set of alternative workflows against a baseline workflow. For both of these two paths we provide an implementation of the diagrams (CD diagrams) described in Demsar (2006) through the functions CDdiagram.BD and CDdiagram.Nemenyi.

The performanceEstimation function ensures that all compared workflows are run on exactly the same train+test partitions on all repetitions and for all predictive tasks. In this context, we can use pairwise statistical significance tests.

Value

The result of this function is the information from performing all these statistical tests. This information is returned as a list with as many components as there are estimated metrics. For each metric a list with several components describing the results of these tests is provided.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Demsar, J. (2006) *Statistical Comparisons of Classifiers over Multiple Data Sets*. Journal of Machine Learning Research, 7, 1-30.

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

CDdiagram.Nemenyi, CDdiagram.BD, signifDiffs, performanceEstimation, topPerformers, topPerformer, rankWorkflows, metricsSummary, ComparisonResults

Examples

```
## Not run:
## Estimating MSE for 3 variants of both
## regression trees and SVMs, on  two data sets, using one repetition
## of 10-fold CV
library(e1071)
data(iris)
data(Satellite,package="mlbench")
data(LetterRecognition,package="mlbench")


## running the estimation experiment
res <- performanceEstimation(
           c(PredTask(Species ~ .,iris),
             PredTask(classes ~ .,Satellite,"sat"),
             PredTask(lettr ~ .,LetterRecognition,"letter")),
           workflowVariants(learner="svm",
                 learner.pars=list(cost=1:4,gamma=c(0.1,0.01))),
           EstimationTask(metrics=c("err","acc"),method=CV()))


## checking the top performers
topPerformers(res)

## now let us assume that we will choose "svm.v2" as our baseline
## carry out the paired comparisons
pres <- pairedComparisons(res,"svm.v2")

## obtaining a CD diagram comparing all others against svm.v2 in terms
## of error rate
CDdiagram.BD(pres,metric="err")


## End(Not run)
```

---

performanceEstimation *Estimate the predictive performance of modeling alternatives on different predictive tasks*

Description

This function can be used to estimate the predictive performance of alternative approaches to a set of predictive tasks, using different estimation methods. This is a generic function that should work with any modeling approaches provided a few assumptions are met. The function implements

different estimation procedures, namely: cross validation, leave one out cross validation, hold-out, monte carlo simulations and bootstrap.

Usage

```
performanceEstimation(tasks,workflows,estTask,...)
```

Arguments

`tasks`	This is a vector of objects of class `PredTask`, containing the predictive tasks that will be used in the estimation procedure.
`workflows`	This is a vector of objects of class `Workflow`, containing the workflows representing different approaches to the predictive tasks, and whose performance we want to estimate.
`estTask`	This is an object belonging to class `EstimationTask`. It is used to specify the metrics to be estimated and the method to use to obtain these estimates. See section Details for the possible values.
`...`	Any further parameters that are to be passed to the lower-level functions implementing each individual estimation methodology.

Details

The goal of this function is to allow estimating the perfomance of a set of alternative modelling approaches on a set of predictive tasks. The estimation can be carried out using different methodologies. All alternative approaches (which we will refer to as *workflows*) will be applied using the same exact data partitions for each task thus ensuring the possibility of carrying out paired comparisons using adequate statistical tests for checking the significance of the observed differences in performance.

The first parameter of this function is a vector of `PredTask` objects that define the tasks to use in the estimation process.

The second argument is a vector of `Workflow` objects. These can be created in two different ways: either directly by calling the constructor of this class; or by using the `workflowVariants` function that can be used to automatically generate different workflow objects as variants of some base workflow. Either way there are two types of workflows: user-defined workflows and what we call "standard" workflows. The later are workflows that people typically follow to solve predictive tasks and that are already implemented in this package to facilitate the task of the user. These standard workflows are implemented in functions `standardWF` and `timeseriesWF`. When specifying the vector of workflows if you use (either in the constructor or in the function `workflowVariants`) the parameter `wf` to indicate which workflow you which to use. If you supply a name different from the two provided standard workflows the function will assume that this is a name of a function you have created to implement your own workflow (see the Examples section for illustrations). In case you omit the value of the `wf` parameter the function assumes you want to use one of the standard workflows and will try to "guess" which one. Namely, if you provide some value for the parameter `type` (either "slide" or "grow"), it will assume that you are addressing a time series task and thus will set `wf` to `timeseriesWF`. In all other cases will set it to `standardWF`. Summarizing, in terms of workflows you can use: i) your own user-defined workflows; ii) the standard workflow implemented by function `standardWF`; or iii) the standard time series workflow implementd by `timeseriesWF`.

Currently, the function allows for 5 different types of estimation methods to be used that are specified when you indicate the esitmation task. These are different methods for providing reliable estimates of the true value of the selected evaluation metrics. Both the metrics and the estimation method are defined through the value provided in argument `estTask`. The 5 estimation methodologies are the following:

Cross validation: this type of estimates can be obtained by providing in `estTask` argument an object of class `EstimationTask` with `method` set to an object of class `CV` (this is the default). More details on this type of method can be obtained in the help page of the class `CV`.

Leave one out cross validation: this type of estimates can be obtained by providing in `estTask` argument an object of class `EstimationTask` with `method` set to an object of class `LOOCV`. More details on this type of method can be obtained in the help page of the class `LOOCV`.

Hold out: this type of estimates can be obtained by providing in `estTask` argument an object of class `EstimationTask` with `method` set to an object of class `Holdout`. More details on this type of method can be obtained in the help page of the class `Holdout`.

Monte Carlo: this type of estimates can be obtained by providing in `estTask` argument an object of class `EstimationTask` with `method` set to an object of class `MonteCarlo`. More details on this type of method can be obtained in the help page of the class `MonteCarlo`.

Bootstrap: this type of estimates can be obtained by providing in `estTask` argument an object of class `EstimationTask` with `method` set to an object of class `Bootstrap`. More details on this type of method can be obtained in the help page of the class `Bootstrap`.

Value

The result of the function is an object of class `ComparisonResults`.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

`workflowVariants`, `topPerformers`, `rankWorkflows`, `pairedComparisons`, `CV`, `LOOCV`, `Holdout`, `MonteCarlo`, `Bootstrap`

Examples

```
## Not run:
## Estimating MSE for 3 variants of both
## regression trees and SVMs, on  two data sets, using one repetition
## of 10-fold CV
library(e1071)
library(DMwR)
data(swiss)
data(mtcars)
```

```
## Estimating MSE using 10-fold CV for 4 variants of a standard workflow
## using an SVM as base learner and 3 variants of a regression tree.
res <- performanceEstimation(
  c(PredTask(Infant.Mortality ~ .,swiss),PredTask(mpg ~ ., mtcars)),
  c(workflowVariants(learner="svm",
                     learner.pars=list(cost=c(1,10),gamma=c(0.01,0.5))),
    workflowVariants(learner="rpartXse",
                     learner.pars=list(se=c(0,0.5,1)))
  ),
  EstimationTask(metrics="mse")
  )

## Check a summary of the results
summary(res)

## best performers for each metric and task
topPerformers(res)


## Estimating the accuracy of a default SVM on IRIS using 10 repetitions
## of a 80%-20% Holdout
data(iris)
res1 <- performanceEstimation(PredTask(Species  ~ .,iris),
             Workflow(learner="svm"),
             EstimationTask(metrics="acc",method=Holdout(nReps=10,hldSz=0.2)))
summary(res1)

## Now an example with a user-defined workflow
myWF <- function(form,train,test,wL=0.5,...) {
    require(rpart,quietly=TRUE)
    ml <- lm(form,train)
    mr <- rpart(form,train)
    pl <- predict(ml,test)
    pr <- predict(mr,test)
    ps <- wL*pl+(1-wL)*pr
    list(trues=responseValues(form,test),preds=ps)
}
resmywf <- performanceEstimation(
             PredTask(mpg ~ ., mtcars),
             workflowVariants(wf="myWF",wL=seq(0,1,by=0.1)),
             EstimationTask(metrics="mae",method=Bootstrap(nReps=50))
           )
summary(resmywf)


## End(Not run)
```

PredTask-class *Class "PredTask"*

Description

This is the class of objects that represent a predictive task

Objects from the Class

Objects can be created by calls to the constructor `PredTask(...)`. The constructor requires a formula and a data frame on the first two arguments. You may also a name for the task through the parameter `taskName` of the constructor. Optional parameter `type` allows you to indicate the type of task (either "regr", "class" or "ts", for regression, classification and time series tasks, respectively). If not provided this will be inferred from constructor. Setting the optional parameter `copy` to `TRUE` (defaults to `FALSE`) will force the constructor to make a copy of the given data frame and store it in the `dataSource` slot of the `PredTask` object.

Slots

`formula`**:** Object of class `formula` containing the formula representing the predictive task.

`dataSource`**:** Object of class `data.frame`, `call` or `name`. This will be used to fecth the task data when necessary. The first of these options will only be used if the user calls the constructor with `copy=TRUE` and will result in the source data being copied into the `PredTask` object.

`taskName`**:** Optional object of class `character` containing the ID of the predictive task

`type`**:** Optional object of class `character` containing the type of the predictive task (it can be "regr", "class" or "ts").

`target`**:** Optional object of class `character` containing the name of the target variable.

Methods

show `signature(object = "PredTask")`: method used to show the contents of a PredTask object.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

`Workflow`, `EstimationTask`, `performanceEstimation`

Examples

```
showClass("PredTask")
data(iris)
PredTask(Species  ~ .,iris)
PredTask(Species  ~ .,iris[1:20,],"irisSubset")
## after the next example you can safely remove the iris data frame from
## your environment because the data was stored inside the "t" object.
t <- PredTask(Species ~ ., iris,copy=TRUE)
```

rankWorkflows *Provide a ranking of workflows involved in an estimation process.*

Description

Given a ComparisonResults object resulting from a performance estimation experiment, this function provides a ranking (by default the top 5) of the *best* workflows involved in the comparison. The rankings are provided by task and for each evaluation metric.

Usage

```
rankWorkflows(compRes,top=min(5,length(workflowNames(compRes))),
         maxs=rep(FALSE,dim(compRes[[1]][[1]]@iterationsScores)[2]),stat="avg")
```

Arguments

compRes	An object of class ComparisonResults with the results of the performance estimation experiment.
top	The number of workflows to include in the rankings (defaulting to 5 or the number of workflows in the experiment if less than 5)
maxs	A vector of booleans with as many elements are there are statistics measured in the experimental comparison. A TRUE value means the respective metric is to be maximized, while a FALSE means minimization. Defaults to all FALSE values.
stat	The statistic to be used to obtain the ranks. The options are the statistics produced by the function summary applied to objects of class ComparisonResults, i.e. "avg", "std", "med", "iqr", "min", "max" or "invalid" (defaults to "avg").

Value

The function returns a named list with as many components as there are predictive tasks in the experiment. For each task you will get another named list, with as many elements as there evaluation metrics. For each of these components you have a data frame with *N* lines, where *N* is the size of the requested rank. Each line includes the name of the workflow in the respective rank position and the estimated score it got on that particular task / evaluation metric.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

performanceEstimation, topPerformers, topPerformer, metricsSummary

Examples

```
## Not run:
## Estimating several evaluation metrics on different variants of a
## regression tree and of a SVM, on  two data sets, using one repetition
## of  10-fold CV

data(swiss)
data(mtcars)
library(e1071)

## run the experimental comparison
results <- performanceEstimation(
               c(PredTask(Infant.Mortality ~ ., swiss),
                 PredTask(mpg ~ ., mtcars)),
               c(workflowVariants(learner='svm',
                                  learner.pars=list(cost=c(1,5),gamma=c(0.1,0.01))
                                 )
               ),
               EstimationTask(metrics=c("mse","mae"),method=CV(nReps=2,nFolds=5))
                                 )
## get a ranking of the top workflows for each task and evaluation metric
rankWorkflows(results)
## get a ranking of the top workflows for each task and evaluation
## metric by the median score on all iterations instead of the mean score
rankWorkflows(results, stat="med")

## End(Not run)
```

`regressionMetrics` *Calculate some standard regression evaluation metrics of predictive performance*

Description

This function is able to calculate a series of regression evaluation statistics given two vectors: one with the true target variable values, and the other with the predicted target variable values. Some of the metrics may require additional information to be given (see Details section).

Usage

```
regressionMetrics(trues, preds, metrics = NULL, train.y = NULL)
```

Arguments

`trues`	A numeric vector with the true values of the target variable.
`preds`	A numeric vector with the predicted values of the target variable.
`metrics`	A vector with the names of the evaluation statistics to calculate (see Details section). If none is indicated (default) it will calculate all available metrics of this function.

`train.y` An optional numeric vector with the values of the target variable on the set of data used to obtain the model whose performance is being tested.

Details

In the following description of the currently available metrics we denote the vector of true target variable values as t, the vector of predictions by p, while N denotes the size of these two vectors, i.e. the number of test cases.

The regression evaluation statistics calculated by this function belong to two different groups of measures: absolute and relative. In terms of absolute error metrics the function includes currently the following:

"mae": mean absolute error, which is calculated as sum(|t_i - p_i|)/N

"mse": mean squared error, which is calculated as sum((t_i - p_i)^2)/N

"rmse": root mean squared error that is calculated as sqrt(mse)

The remaining measures ("mape", "nmse", "nmae" and "theil") are relative measures, the three later comparing the performance of the model with a baseline. They are unit-less measures with values always greater than 0. In the case of "nmse", "nmae" and "theil" the values are expected to be in the interval [0,1] though occasionaly scores can overcome 1, which means that your model is performing worse than the baseline model. The baseline used in both "nmse" and "nmae" is a constant model that always predicts the average target variable value, estimated using the values of this variable on the training data (data used to obtain the model that generated the predictions), which should be provided in the parameter `train.y`. The "theil" metric is typically used in time series tasks and the used baseline is the last observed value of the target variable. The relative error measure "mape" does not require a baseline. It simply calculates the average percentage difference between the true values and the predictions.

These measures are calculated as follows:

"mape": sum(|(t_i - p_i) / t_i|)/N

"nmse": sum((t_i - p_i)^2) / sum((t_i - AVG(Y))^2), where AVG(Y) is the average of the values provided in vector `train.y`

"nmae": sum(|t_i - p_i|) / sum(|t_i - AVG(Y)|)

"theil": sum((t_i - p_i)^2) / sum((t_i - t_{i-1})^2), where t_{i-1} is the last observed value of the target variable

The user may also indicate the value "all" in the parameter `metrics`. In this case all possible metrics will be calculated. This will only include the "nmse", "nmae" and "theil" metrics if the value of the `train.y` parameter is set, otherwise only the other metrics will be returned.

Value

A named vector with the calculated evaluation scores.

Note

In case you require either "nmse", "nmae" or "theil" to be calculated you must supply a vector of numeric values through the parameter `train.y`, otherwise the function will return an error message. These values are required to obtain a fair baseline against which your model predictions will be compared to.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

classificationMetrics

Examples

```
## Not run:
## Calculating several statistics of a regression tree on the Swiss data
data(swiss)
idx <- sample(1:nrow(swiss),as.integer(0.7*nrow(swiss)))
train <- swiss[idx,]
test <- swiss[-idx,]
library(rpart)
model <- rpart(Infant.Mortality ~ .,train)
preds <- predict(model,test)
## by default only mse is calculated
regressionMetrics(test[,'Infant.Mortality'],preds)
## calculate mae and rmse
regressionMetrics(test[,'Infant.Mortality'],preds,metrics=c('mae','rmse'))
## calculate all statistics
regressionMetrics(test[,'Infant.Mortality'],preds,train.y=train[,'Infant.Mortality'])

## End(Not run)
```

---

responseValues *Obtain the target variable values of a prediction task*

Description

This function obtains the values in the column whose name is the target variable of a prediction problem described by a formula.

Usage

```
responseValues(formula, data, na=NULL)
```

Arguments

formula	A formula describing a prediction problem
data	The data frame containing the data of the prediction problem
na	A function to handle NAs in the data

Value

A vector of values

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

Examples

```
data(iris)
tgt <- responseValues(Species ~ ., iris)
summary(tgt)
```

---

results2table *Obtains a dplyr data frame table object containing all the results of an experiment*

Description

This function produces a dplyr data frame table object with the information on all iterations of an experiment. This type of objects may be easier to manipulate in terms of querying these results, particular for larger experiments involving lots of tasks, workflows and metrics.

Usage

```
results2table(res)
```

Arguments

res	This is a ComparisonResults object (type "class?ComparisonResults" for details) that contains the results of a performance estimation experiment obtained through the performanceEstimation() function.

Value

The function returns a dplyr data frame table object containing all resutls of the experiment. The object has the columns: Task, Workflow, nrIt, Metric and Score. Each row is one train+test cycle within the experiment, i.e. contains the score of some metric obtained by some workflow on one train+test iteration of a task.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

getScores, performanceEstimation

Examples

```
## Not run:
## Estimating MSE for 3 variants of both
## regression trees and SVMs, on  two data sets, using one repetition
## of 10-fold CV
library(e1071)
data(swiss)

## running the estimation experiment
res <- performanceEstimation(
  PredTask(Infant.Mortality ~ .,swiss,"Swiss"),
  workflowVariants(learner="svm",
                   learner.pars=list(cost=c(1,10),gamma=c(0.01,0.5))),
  EstimationTask(metrics=c("mse","nmae"),method=CV(nReps=2,nFolds=5))
  )

## Obtaining a table with the results
library(dplyr)
tbl <- results2table(res)

## Mean and standard deviation of each workflow per task (only one in
## this example) and metric
group_by(tbl,Task,Workflow,Metric)
     summarize_each_(funs(mean,sd),"Score")

## Top 2 workflows in terms of MSE for this task
filter(tbl,Task=="Swiss",Metric=="mse")
    group_by(Workflow)
      summarize_each_(funs(mean),"Score")
        arrange(Score)
          slice(1:2)

## End(Not run)
```

---

runWorkflow *Run a workflow on a predictive task*

Description

This function can be used to apply (run) a workflow.

It receives in the first argument an object of class Workflow and then any other arguments that are required to run the workflow, which will typically involve at least a formula, a training data set (a data frame), and a test data set (another data frame).

Usage

```
runWorkflow(l, ...)
```

Arguments

l	An object of class Workflow
...	Further arguments that are required by the workflow function

Value

The execution of a workflow should produce a list as result. If you plan to apply any of the functions provided in this package to calculate standard performance metrics (classificationMetrics or regressionMetrics) then your list should contain at least two components: one named *trues* with the true values of the target variable in the test set, and the other named *preds* with the respective predictions of the workflow.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

Workflow

Examples

```
## Not run:
## A simple example
data(iris)
w1 <- Workflow("mySolution",par1="gaussian",k=10)
runWorkflow(w1,Species ~ .,iris[1:100,],iris[101:150,])

## End(Not run)
```

signifDiffs *Obtains a list with the set of paired differences that are statistically significant according to a p-value threshold*

Description

This function receives as main argument the object resulting from a call to the pairedComparisons function and produces a list with the subset of the paired comparisons using the *t* test and the *Wilcoxon Signed Rank* test that are statistically significant given a certain *p* value limit.

Usage

```
signifDiffs(ps, p.limit=0.05,
            metrics=names(ps),
            tasks=rownames(ps[[1]]$avgScores))
```

Arguments

ps	An object resulting from a call to the pairedComparisons function.
p.limit	A number indicating the maximum value of the confidence level (p.value) of the statistical hypothesis test for a paired comparison to be considered statistically significant (defaults to 0.05). All paired comparisons with a *p* value below this limit will appear in the results of this function.
metrics	A vector with the names of the metrics for which we want the results (defaults to all metrics included in the paired comparison).
tasks	A vector with the names of the prediction tasks for which we want the results (defaults to all tasks included in the paired comparison).

Details

This function produces a list with as many components as the selected metrics (defaulting to all metrics in the paired comparison). Each of the components is another list with two components: i) one with the results for the *t* tests; and ii) the other with the results for the *Wilcoxon Signed Rank* test. Each of these two components is an array with 3 dimensions, with the rows representing the workflows, the columns a set of statistics and the thrid dimension being the task. The first row of these arrays will contain the baseline workflow against which all others are being compared (by either the *t* test or through the *Wilcoxon Signed Rank* test). The remaining rows will include the workflows whose comparison against this baseline is statistically significant, i.e. whose *p* value of the paired comparison is below the provided *p* limit.

Value

The result of this function is a list (see the Details section).

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

pairedComparisons, performanceEstimation

Examples

```
## Not run:
## Estimating MSE for 3 variants of both
## regression trees and SVMs, on  two data sets, using one repetition
## of 10-fold CV
library(e1071)
data(iris)
data(Satellite,package="mlbench")
data(LetterRecognition,package="mlbench")


## running the estimation experiment
res <- performanceEstimation(
           c(PredTask(Species ~ .,iris),
             PredTask(classes ~ .,Satellite,"sat"),
             PredTask(lettr ~ .,LetterRecognition,"letter")),
           workflowVariants(learner="svm",
                 learner.pars=list(cost=1:4,gamma=c(0.1,0.01))),
           EstimationTask(metrics=c("err","acc"),method=CV()))

## now let us assume that we will choose "svm.v2" as our baseline
## carry out the paired comparisons
pres <- pairedComparisons(res,"svm.v2")

## Obtaining the subset of differences that are significant
## with 99% confidence
sds <- signifDiffs(res,p.limit=0.01)


## End(Not run)
```

---

smote *SMOTE algorithm for unbalanced classification problems*

Description

This function handles unbalanced classification problems using the SMOTE method. Namely, it can generate a new "SMOTEd" data set that addresses the class unbalance problem.

Usage

```
smote(form, data, perc.over = 2, k = 5, perc.under = 2)
```

Arguments

form	A formula describing the prediction problem
data	A data frame containing the original (unbalanced) data set
perc.over	A number that drives the decision of how many extra cases from the minority class are generated (known as over-sampling).
k	A number indicating the number of nearest neighbours that are used to generate the new examples of the minority class.
perc.under	A number that drives the decision of how many extra cases from the majority classes are selected for each case generated from the minority class (known as under-sampling)

Details

Unbalanced classification problems cause problems to many learning algorithms. These problems are characterized by the uneven proportion of cases that are available for each class of the problem.

SMOTE (Chawla et. al. 2002) is a well-known algorithm to fight this problem. The general idea of this method is to artificially generate new examples of the minority class using the nearest neighbors of these cases. Furthermore, the majority class examples are also under-sampled, leading to a more balanced dataset.

The parameters perc.over and perc.under control the amount of over-sampling of the minority class and under-sampling of the majority classes, respectively. perc.over will tipically be a number above 1. With this type of values, for each case in the orginal data set belonging to the minority class, perc.over new examples of that class will be created. If perc.over is a value below 1 than a single case will be generated for a randomly selected proportion (given by perc.over) of the cases belonging to the minority class on the original data set. The parameter perc.under controls the proportion of cases of the majority class that will be randomly selected for the final "balanced" data set. This proportion is calculated with respect to the number of newly generated minority class cases. For instance, if 200 new examples were generated for the minority class, a value of perc.under of 1 will randomly select exactly 200 cases belonging to the majority classes from the original data set to belong to the final data set. Values above 1 will select more examples from the majority classes.

The parameter k controls the way the new examples are created. For each currently existing minority class example X new examples will be created (this is controlled by the parameter perc.over as mentioned above). These examples will be generated by using the information from the k nearest neighbours of each example of the minority class. The parameter k controls how many of these neighbours are used.

Value

The function returns a data frame with the new data set resulting from the application of the SMOTE algorithm.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). *Smote: Synthetic minority over-sampling technique*. Journal of Artificial Intelligence Research, 16:321-357.

Torgo, L. (2010) *Data Mining using R: learning with case studies*, CRC Press (ISBN: 9781439810187). http://www.dcc.fc.up.pt/~ltorgo/DataMiningWithR

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

Examples

```
## A small example with a data set created artificially from the IRIS
## data
data(iris)
data <- iris[, c(1, 2, 5)]
data$Species <- factor(ifelse(data$Species == "setosa","rare","common"))
## checking the class distribution of this artificial data set
table(data$Species)

## now using SMOTE to create a more "balanced problem"
newData <- smote(Species ~ ., data, perc.over = 6,perc.under=1)
table(newData$Species)

## Checking visually the created data
## Not run:
par(mfrow = c(1, 2))
plot(data[, 1], data[, 2], pch = 19 + as.integer(data[, 3]),
     main = "Original Data")
plot(newData[, 1], newData[, 2], pch = 19 + as.integer(newData[,3]),
     main = "SMOTE'd Data")

## End(Not run)
```

`standardPOST` *A function for applying post-processing steps to the predictions of a model*

Description

This function implements a series of simple post-processing steps to be applied to the predictions of a model. It also allows the user to supply hers/his own post-processing functions. The result of the function is a new version of the predictions of the model (typically a vector or a matrix in the case of models that predict class probabilities, for instance).

Usage

```
standardPOST(form, train, test, preds, steps, ...)
```

Arguments

form	A formula specifying the predictive task.
train	A data frame containing the training set.
test	A data frame containing the test set.
preds	The object resulting from the application of a model to the test set to obtain its predictions (typically a vector or a matrix for probabilistic classifiers)
steps	A vector with function names that are to be applied in the sequence they appear in this vector to the predictions to obtain a new version of these predictions.
...	Any further parameters that will be passed to all functions specified in steps

Details

This function is mainly used by both standardWF and timeseriesWF as a means to allow for users of these two standard workflows to specify some post-processing steps for the predictions of the models. These are steps one wishes to apply to the predictions to somehow change the outcome of the prediction stage.

Nevertheless, the function can also be used outside of these standard workflows for obtaining post-processed versions of the predictions.

The function accepts as post-processing functions both some already implemented functions as well as any function defined by the user provided these follow some protocol. Namely, these user-defined post-processing functions should be aware that they will be called with a formula, a training data frame, a testing data frame and the predictions in the first four arguments. Moreover, any arguments used in the call to standardPOST will also be forwarded to these user-defined functions. Finally, these functions should return a new version of the predictions. It is questionable the interest of supplying both the training and test sets to these functions, on top of the formula and the predictions. However, we have decided to pass them anyway not to precule the usage of any special post-processing step that requires this information.

The function already contains implementations of the following post-processing steps that can be used in the steps parameter:

"na2central" - this function fills in any NA predictions into either the median (numeric targets) or mode (nominal targets) of the target variable on the training set. Note that this is only applicable to predictions that are vectors of values.

"onlyPos" - in some numeric forecasting tasks the target variable takes only positive values. Nevertheless, some models may insist in forecasting negative values. This function casts these negative values to zero. Note that this is only applicable to predictions that are vectors of numeric values.

"cast2int" - in some numeric forecasting tasks the target variable takes only values within some interval. Nevertheless, some models may insist in forecasting values outside of this interval. This function casts these values into the nearest interval boundary. This function requires that you supply the limits of this interval through parameters infLim and supLim. Note that this is only applicable to predictions that are vectors of numeric values.

"maxutil" - maximize the utility of the predictions (Elkan, 2001) of a classifier. This method is only applicable to classification tasks and to algorithms that are able to produce as predictions a vector of class probabilities for each test case, i.e. a matrix of probabilities for a given test set. The method requires a cost-benefit matrix to be provided through the parameter `cb.matrix`. For each test case, and given the probabilities estimated by the classifier and the cost benefit matrix, the method predicts the classifier that maximizes the utility of the prediction. This approach (Elkan, 2001) is a slight 'evolution' of the original idea (Breiman et al., 1984) that only considered the costs of errors and not the benefits of the correct classifications as in the case of cost-benefit matrices we are using here. The parameter `cb.matrix` must contain a (square) matrix of dimension NClasses x NClasses where entry X_i,j corresponds to the cost/benefit of predicting a test case as belonging to class j when it is of class i. The diagonal of this matrix (correct predicitons) should contain positive numbers (benefits), whilst numbers outside of the matrix should contain negative numbers (costs of misclassifications). See the Examples section for an illustration.

Value

An object of the same class as the input parameter `preds`

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Breiman,L., Friedman,J., Olshen,R. and Stone,C. (1984), *Classification and Regression Trees*, Wadsworth and Brooks.

Elkan, C. (2001), *The Foundations of Cost-Sensitive Learning*. Proceedings of IJCAI'2001.

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

standardPRE, standardWF, timeseriesWF

Examples

```
## Not run:

######
## Using in the context of an experiment


data(algae,package="DMwR")
library(e1071)

## This will issue several warnings because this implementation of SVMs
## will ignore test cases with NAs in some predictor. Our infra-structure
## issues a warning and fills in these with the prediction of an NA
res <- performanceEstimation(
  PredTask(a1 ~ .,algae[,1:12],"alga1"),
```

```
  Workflow(learner="svm"),
  EstimationTask(metrics="mse")
  )
summary(getIterationPreds(res,1,1,it=1))

## one way of overcoming this would be to post-process the NA
## predictions into a statistic of centrality
resN <- performanceEstimation(
  PredTask(a1 ~ .,algae[,1:12],"alga1"),
  Workflow(learner="svm",post="na2central"),
  EstimationTask(metrics="mse")
  )
summary(getIterationPreds(resN,1,1,it=1))

## because the SVM also predicts negative values which does not make
## sense in this application (the target are frequencies thus >= 0) we
## could also include some further post-processing to take care of
## negative predictions
resN <- performanceEstimation(
  PredTask(a1 ~ .,algae[,1:12],"alga1"),
  Workflow(learner="svm",post=c("na2central","onlyPos")),
  EstimationTask(metrics="mse")
  )
summary(getIterationPreds(resN,1,1,it=1))

######################
## An example with utility maximization learning for the
## BreastCancer data set on package mlbench
##
data(BreastCancer,package="mlbench")

## First lets create the cost-benefit matrix
cb <- matrix(c(1,-10,-100,100),byrow=TRUE,ncol=2)
colnames(cb) <- paste("p",levels(BreastCancer$Class),sep=".")
rownames(cb) <- paste("t",levels(BreastCancer$Class),sep=".")

## This leads to the following cost-benefit matrix
##             p.benign p.malignant
## t.benign           1         -10
## t.malignant     -100         100

## Now the performance estimation. We are estimating error rate (wrong
## for cost sensitive tasks!) and total utility of the model predictions
## (the right thing to do here!)
library(rpart)

r <- performanceEstimation(
        PredTask(Class ~ .,BreastCancer[,-1],"breastCancer"),
        c(Workflow(wfID="rpart.cost",
                   learner="rpart",
                   post="maxutil",
                   post.pars=list(cb.matrix=cb)
                  ),
```

```
             Workflow(wfID="rpart",
                      learner="rpart",
                      predictor.pars=list(type="class")
                     )
             ),
            EstimationTask(
                 metrics=c("err","totU"),
                 evaluator.pars=list(benMtrx=cb,posClass="malignant"),
                 method=CV(strat=TRUE)))

## Analysing the results
rankWorkflows(r,maxs=c(FALSE,TRUE))

## Visualizing them
plot(r)


## End(Not run)
```

`standardPRE` *A function for applying data pre-processing steps*

Description

This function implements a series of simple data pre-processing steps and also allows the user to supply hers/his own functions to be applied to the data. The result of the function is a list containing the new (pre-processed) versions of the given train and test sets.

Usage

```
standardPRE(form, train, test, steps, ...)
```

Arguments

`form`	A formula specifying the predictive task.
`train`	A data frame containing the training set.
`test`	A data frame containing the test set.
`steps`	A vector with function names that are to be applied in the sequence they appear in this vector to both the training and testing sets, to obtain new versions of these two data samples.
`...`	Any further parameters that will be passed to all functions specified in `steps`

Details

This function is mainly used by both standardWF and timeseriesWF as a means to allow for users of these two standard workflows to specify some data pre-processing steps. These are steps one wishes to apply to the different train and test samples involved in an experimental comparison, before any model is learned or any predictions are obtained.

Nevertheless, the function can also be used outside of these standard workflows for obtaining pre-processed versions of train and test samples.

The function accepts as pre-processing functions both some already implemented functions as well as any function defined by the user provided these follow some protocol. Namely, these user-defined pre-processing functions should be aware that they will be called with a formula, a training data frame and a testing data frame in the first three arguments. Moreover, any arguments used in the call to standardPRE will also be forwarded to these user-defined functions. Finally, these functions should return a list with two components: "train" and "test", containing the pre-processed versions of the supplied train and test data frames.

The function already contains implementations of the following pre-processing steps that can be used in the steps parameter:

"scale" - that scales (subtracts the mean and divides by the standard deviation) any knnumeric features on both the training and testing sets. Note that the mean and standard deviation are calculated using only the training sample.

"centralImp" - that fills in any NA values in both sets using the median value for numeric predictors and the mode for nominal predictors. Once again these centrality statistics are calculated using only the training set although they are applied to both train and test sets.

"knnImp" - that fills in any NA values in both sets using the median value for numeric predictors and the mode for nominal predictors, but using only the k-nearest neighbors to calculate these satistics.

"na.omit" - that uses the R function na.omit to remove any rows containing NA's from both the training and test sets.

"undersampl" - this undersamples the training data cases that do not belong to the minority class (this pre-processing step is only available for classification tasks!). It takes the parameter perc.under that controls the level of undersampling (defaulting to 1, which means that there would be as many cases from the minority as from the other(s) class(es)).

"smote" - this operation uses the SMOTE (Chawla et. al. 2002) resampling algorithm to generate a new training sample with a more "balanced" distributions of the target class (this pre-processing step is only available for classification tasks!). It takes the parameters perc.under, perc.over and k to control the algorithm. Read the documentation of function smote to know more details.

Value

A list with components "train" and "test" with both containing a data frame.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). *Smote: Synthetic minority over-sampling technique*. Journal of Artificial Intelligence Research, 16:321-357.

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

standardPOST, standardWF, timeseriesWF

Examples

```
## Not run:

##  A small example with standard pre-preprocessing: clean NAs and scale
data(algae,package="DMwR")

idx <- sample(1:nrow(algae),150)
tr <- algae[idx,1:12]
ts <- algae[-idx,1:12]
summary(tr)
summary(ts)

preData <- standardPRE(a1 ~ ., tr, ts, steps=c("centralImp","scale"))
summary(preData$train)
summary(preData$test)

######
## Using in the context of an experiment
library(e1071)
res <- performanceEstimation(
  PredTask(a1 ~ .,algae[,1:12],"alga1"),
  Workflow(learner="svm",pre=c("centralImp","scale")),
  EstimationTask(metrics="mse")
  )

summary(res)

######
## A user-defined pre-processing function
myScale <- function(f,tr,ts,avg,std,...) {
    tgtVar <- deparse(f[[2]])
    allPreds <- setdiff(colnames(tr),tgtVar)
    numPreds <- allPreds[sapply(allPreds,
                          function(p) is.numeric(tr[[p]]))]
    tr[,numPreds] <- scale(tr[,numPreds],center=avg,scale=std)
    ts[,numPreds] <- scale(ts[,numPreds],center=avg,scale=std)
    list(train=tr,test=ts)
}

## now using it with some random averages and stds for the 8 numeric
```

```
## predictors (just for illustration)
newData <- standardPRE(a1 ~ .,tr,ts,steps="myScale",
                       avg=rnorm(8),std=rnorm(8))


## End(Not run)
```

`standardWF` *A function implementing a standard workflow for prediction tasks*

Description

This function implements a standard workflow for both classification and regression tasks. The workflow consists of: (i) learning a predictive model based on the given training set, (ii) using it to make predictions for the provided test set, and finally (iii) measuring some evaluation metrics of its performance.

Usage

```
standardWF(form,train,test,
    learner,learner.pars=NULL,
    predictor='predict',predictor.pars=NULL,
    pre=NULL,pre.pars=NULL,
    post=NULL,post.pars=NULL,
    .fullOutput=FALSE)
```

Arguments

`form`	A formula specifying the predictive task.
`train`	A data frame containing the data set to be used for obtaining the predictive model (the training set).
`test`	A data frame containing the data set to be used for testing the obtained model (the test set).
`learner`	A character string with the name of a function that is to be used to obtain the prediction models.
`learner.pars`	A list of parameter values to be passed to the learner (defaults to `NULL`).
`predictor`	A character string with the name of a function that is to be used to obtain the predictions for the test set using the obtained model (defaults to 'predict').
`predictor.pars`	A list of parameter values to be passed to the predictor (defaults to `NULL`).
`pre`	A vector of function names that will be applied in sequence to the train and test data frames, generating new versions, i.e. a sequence of data pre-processing functions.
`pre.pars`	A named list of parameter values to be passed to the pre-processing functions.

post	A vector of function names that will be applied in sequence to the predictions of the model, generating a new version, i.e. a sequence of data post-processing functions.
post.pars	A named list of parameter values to be passed to the post-processing functions.
.fullOutput	A boolean that if set to TRUE will make the function return more information in the list that results from its execution like for instance the models (defaults to FALSE).

Details

The main goal of this function is to facilitate the task of the users of the experimental comparison infra-structure provided by function performanceEstimation. Namely, this function requires the users to specify the workflows (solutions to predictive tasks) whose performance she/he wants to estimate and compare. The user has the flexibility of writing hers/his own workflow functions, however, in most situations that is not really necessary. The reason is that most of the times users just want to compare standard out of the box learning algorithms on some tasks. In these contexts, the workflow simply consists of applying some existing learning algorithm to the training data, and then use it to obtain the predictions of the test set. This standard workflow may even include some standard pre-processing tasks applied to the given data before the model is learned, and eventually some post processing tasks applied to the predictions before they are returned to the user. The goal of the current function is to facilitate evaluating this sort of estimation experiments. It implements this workflow thus avoiding the need of the user to write these workflows.

Through parameter learner users may indicate the modeling algorithm to use to obtain the predictive model. This can be any R function, provided it can be called with a formula on the first argument and a training set on a parameter named data (as most R modeling functions do). As usual, these functions may include other arguments that are specific to the modeling approach (i.e. are parameters of the model). The values to be used for these parameters are specified as a list through the parameter learner.pars of function standardWF. The learning function can return any class of object that represents the learned model. This object will be used to obtain the predictions in this standard workflow.

Equivalently, the user may specify a function for obtaining the predictions for the test set using the previously learned model. Again this can be any function, and it is indicated in parameter predictor (defaulting to the usual predict function). This function should be prepared to accept in the first argument the learned model and in the second the test set, and should return the predictions of the model for this set of data. It may also have additional parameters whose values are specified as a list in parameter predictor.pars.

Additionally, the user may specify a set of data-preprocessing functions to be applied to both the training and testing sets, through parameter pre that accepts a vector of function names. These functions will be applied to both the training and testing sets, in the sequence provided in the vector of names, before the learning algorithm is applied to the training set. Once again the user is free to indicate as pre-processing functions any function, eventually her/his own functions carrying our any sort of pre-processing steps. These user-defined pre-processing functions will be applied by function standardPRE. Check the help page of this function to know the protocol you need to follow to be able to use your own pre-processing functions. Still, our infra-structure already includes some common pre-processing functions so that you do not need to implement them. The list of these functions is again described in the help page of standardPRE.

The predictions obtained by the function specified in parameter `predict` may also go through some post-processing steps before they are return as a result of the `standardWF` function. Again the user may specify a vector of post-processing functions to be applied in sequence, through the parameter `post`. Parameters to be passed to these functions can be specified through the parameter `post.pars`. The goal of these functions is to obtain a new version of the predictions of the models after going through some post-processing steps. These functions will be applied to the predictions by the function `standardPOST`. Once again this function already implements a few standard post-processing steps but you are free to supply your own post-processing functions provided they follow the protocol described in the help page of function `standardPOST`.

Finally, the parameter `.fullOutput` controls the ammount of information that is returned by the `standardWF` function. By default it is `FALSE` which means that the workflow will only return (apart from the predictions) the train, test and total times of the learning and prediction stages. This information is returned as a component named "times" of the results list that can be obtained for instance by using the `getIterationsInfo` if the workflow is being used in the context of an experimental comparison. If `.fullOutput` is set to `TRUE` the workflow will also include information on the pre-processing steps (in a component named "preprocessing"), information on the model and predictions of the model (in a component named "modeling") and information on the post-processing steps (in a component named "postprocessing").

Value

A list with several components containing the result of runing the workflow.

Note

In order to use any of the available learning algorithms in R you must have previously installed and loaded the respective packages, if necessary.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

`performanceEstimation`, `timeseriesWF`, `getIterationsInfo`, `getIterationsPreds`, `standardPRE`, `standardPOST`

Examples

```
## Not run:
data(iris)
library(e1071)

## a standard workflow using and SVM with default parameters
w.s <- Workflow(wfID="std.svm",learner="svm")
```

```
w.s

irisExp <- performanceEstimation(
  PredTask(Species ~ .,iris),
  w.s,
  EstimationTask("acc"))

getIterationsPreds(irisExp,1,1,it=4)
getIterationsInfo(irisExp,1,1,rep=1,fold=2)

## A more sophisticated standardWF
## - as pre-processing we imput NAs with either the median (numeric
## features) or the mode (nominal features); and we also scale
## (normalize) the numeric predictors
## - as learning algorithm we use and SVM with cost=10 and gamma=0.01
## - as post-processing we scale all predictions into the range [0..50]
w.s2 <- Workflow(pre=c("centralImp","scale"),
                 learner="svm",
                 learner.pars=list(cost=10,gamma=0.01),
                 post="cast2int",
                 post.pars=list(infLim=0,supLim=50),
                 .fullOutput=TRUE
                )

data(algae,package="DMwR")

a1.res <- performanceEstimation(
            PredTask(a1 ~ ., algae[,1:12],"alga1"),
            w.s2,
            EstimationTask("mse")
            )

## Workflow variants of a standard workflow
ws <- workflowVariants(
                 pre=c("centralImp","scale"),
                 learner="svm",
                 learner.pars=list(cost=c(1,5,10),gamma=0.01),
                 post="cast2int",
                 post.pars=list(infLim=0,supLim=c(10,50,80)),
                 .fullOutput=TRUE,
                 as.is="pre"
                )
a1.res <- performanceEstimation(
            PredTask(a1 ~ ., algae[,1:12],"alga1"),
            ws,
            EstimationTask("mse")
            )

## An example using GBM that is a bit different in terms of the
## prediction part as it requires to select the number of trees of the
## ensemble to use
data(Boston, package="MASS")
library(gbm)
```

```
## A user written predict function to allow for using the standard
## workflows
gbm.predict <- function(model, test, method, ...) {
    best <- gbm.perf(model, plot.it=FALSE, method=method)
    return(predict(model, test, n.trees=best, ...))
}

resG <- performanceEstimation(
             PredTask(medv ~.,Boston),
             Workflow(learner="gbm",
                      learner.pars=list(n.trees=1000, cv.folds=10),
                      predictor="gbm.predict",
                      predictor.pars=list(method="cv")),
             EstimationTask(metrics="mse",method=CV())
        )


## End(Not run)
```

`subset-methods` *Methods for Function* `subset` *in Package* **performanceEstimation**

Description

The method subset when applied to objects of class `ComparisonResults` can be used to obtain another object of the same class with a subset of the estimation results contained in the original object.

Methods

`signature(x = "ComparisonResults")` The method has as first argument the object of class `ComparisonResults` that you wish to subset. This method also includes 4 extra arguments that you can use to supply the subsetting criteria.

Namely, the parameter `metrics` allows you to indicate a vector with the subset of evaluation metrics in the orginal object. Alternatively, you can provide a regular expression to be matched against the name of the statistics measured in the experiment (see function `metricNames`) to specify the subset of metrics you want to select.

The parameter `workflows` can be used to provide a vector with the subset of workflows (approaches to the predictive tasks) that are to be used in the sub-setting. Alternatively, you can also provide a regular expression to be matched against the name of the workflows (see function `workflowNames`) evaluated in the experiment to specify the subset you want to select.

The parameter `tasks` allows you to indicate a vector with the subset of predictive tasks to be extracted. Alternatively, you can provide a regular expression to be matched against the name of the tasks used in the experiment (see function `taskNames`) to specify the subset you want to select.

Finally, the parameter `partial` allows you to control how the names of the other parameters (tasks, workflows and metrics) are matched against the original names in the `ComparisonResults`

object. It defaults to TRUE which means that the function grep is used, whilst with FALSE the mathcin is done using the function match. For instance, if you have three metrics being estimated with names "F", "F1" and "F2", and you call subset with metrics="F", this would in effect subset all three metrics, whilst with partial=FALSE only the first of the three would be considered.

---

taskNames *The prediction tasks involved in an estimation experiment*

Description

This function can be used to get a vector with the IDs of the prediction tasks that were used in a performance estimation experiment.

Usage

```
taskNames(o)
```

Arguments

o	An object of class ComparisonResults

Value

A vector of strings (the names of the tasks)

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

ComparisonResults, performanceEstimation, metricNames, workflowNames

Examples

```
## Not run:
## Estimating MSE for 3 variants of both
## regression trees and SVMs, on  two data sets, using one repetition
## of 10-fold CV
library(e1071)
library(DMwR)
data(swiss)
data(mtcars)
```

```
## running the estimation experiment
res <- performanceEstimation(
  c(PredTask(Infant.Mortality ~ .,swiss),PredTask(mpg ~ ., mtcars)),
  c(workflowVariants(learner="svm",
                     learner.pars=list(cost=c(1,10),gamma=c(0.01,0.5))),
    workflowVariants(learner="rpartXse",
                     learner.pars=list(se=c(0,0.5,1)))
  ),
  EstimationTask("mse")
  )

## the names of the metrics that were estimated in the above experiment
taskNames(res)

## End(Not run)
```

`timeseriesWF` *A function implementing sliding and growing window standard workflows for time series forecasting tasks*

Description

This function implements sliding and growing window workflows for the prediction time series. The sliding window workflow consists of: (i) starting by learning a prediction model based on the given training set, (ii) use this model to obtain predictions for a pre-defined number of future time steps of the test set; (iii) then slide the training window forward this pre-defined number of steps and obtain a new model with this new training set; (iv) use this new model for obtaining another set of predictions; and (v) keep repeting this sliding process until predictions are obtained for all test set period.

The growing window workflow is similar but instead of sliding the training window, we grow this window, so each new set of predictions is obtained with a model learned with all data since the beginning of the training set till the current time step.

Usage

```
timeseriesWF(form,train,test,
   learner,learner.pars=NULL,
   type='slide',relearn.step=1,
   predictor='predict',predictor.pars=NULL,
   pre=NULL,pre.pars=NULL,
   post=NULL,post.pars=NULL,
   .fullOutput=FALSE,verbose=FALSE)
```

Arguments

`form`	A formula specifying the predictive task.

train	A data frame containing the data set to be used for obtaining the first prediction model. In case we are using the sliding window approach, the size of this training set will determine the size of all future training sets after each slide step.
test	A data frame containing the data set for which we want predictions.
learner	A character string with the name of a function that is to be used to obtain the prediction models.
learner.pars	A list of parameter values to be passed to the learner (defaults to NULL).
type	A character string specifying if we are using a sliding (value 'slide') or a growing (value 'grow') window workflow (defaults to 'slide').
relearn.step	The number of time steps (translated into number of rows in the test set) after which a new model is re-learned (either by sliding or growing the training window) (defaults to 1, i.e. each new row).
predictor	A character string with the name of a function that is to be used to obtain the predictions for the test set using the obtained model (defaults to 'predict').
predictor.pars	A list of parameter values to be passed to the predictor (defaults to NULL).
pre	A vector of function names that will be applied in sequence to the train and test data frames, generating new versions, i.e. a sequence of data pre-processing functions.
pre.pars	A named list of parameter values to be passed to the pre-processing functions.
post	A vector of function names that will be applied in sequence to the predictions of the model, generating a new version, i.e. a sequence of data post-processing functions.
post.pars	A named list of parameter values to be passed to the post-processing functions.
.fullOutput	A boolean that if set to TRUE will make the function return more information in the list that results from its execution like for instance the models (defaults to FALSE).
verbose	A Boolean indicating whether a "*" character should be printed every time the window slides (defaults to FALSE).

Details

The main goal of this function is to facilitate the task of the users of the experimental comparison infra-structure provided by function performanceEstimation for time series problems where the target variable can be numeric or nominal. Frequently, users just want to compare existing algorithms or variants of these algorithms on a set of forecasting tasks, using some standard error metrics. The goal of the timeseriesWF function is to facilitate this task by providing a standard workflow for time series tasks.

The function works, and has almost the same parameters, as function standardWF. The help page of this latter function describes most of the parameters used in the current function and thus we will not repeat the description here. The main difference to the standardWF function is on the existance of two extra parameters that control the sliding and growing window approaches to time series forecasting. These are parameters type and relearn.step. We have considered two typical workflow approaches for time series tasks where the user wants predictions for a certain future time period. Both are based on the assumption that after "some" time the model that we have obtained

with the given training period data may have become out-dated, and thus a new model should be obtained with the most recent data. The idea is that as we move in the testing period and get predictions for the successive rows of the test set, it is like if a clock is advancing. Previous rows for which we already made a prediction are "past" as we assume that the successive rows in both the `train` and `test` data frames are ordered by time (they are time series). In this context, as we move forward in the test period we can regard the rows for which we already made a prediction as past data, and thus potentially useful to be added to our initial training set with the goal of obtaining a fresh new model with more recent data. This type of reasoning only makes sense if we suspect that there is some kind of concept drift on our data. For stationary data this makes no sense and we would be better off using the workflow provided by function `standardWF`. Still, the current function implements two workflows following this model-updating reasoning: (i) sliding window; and (ii) growing window. Both use the value of the parameter (`relearn.step`) to decide the number of time periods after which we re-learn the model using fresh new data. The difference between the two strategies lies on how they treat the oldest data (the initial rows of the provided training set). Sliding window, as the name suggests, after each relearn step slides the training set forward thus forgetting the oldest rows of the previous training set whilst incorporating the most recent observations. With this approach all models are obtained with a training set with the same amount of data (the number of rows of the initially given training set). Growing window does not remove older rows and thus the training sets keep growing in size after each relearn step.

Value

A list with several components containing the result of runing the workflow.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

`standardWF`, `performanceEstimation`, `getIterationsInfo`, `getIterationsPreds`, `standardPRE`, `standardPOST`

Examples

```
## The following is a small illustrative example using the quotes of the
## SP500 index. This example compares two random forests with 500
## regression trees, one applyed in a standard way, and the other using
## a sliding window with a relearn step of every 10 days. The experiment
## uses 10 repetitions of a train+test cycle using 50% of the available
## data for training and 25% for testing.
## Not run:
library(quantmod)
library(randomForest)
```

```
getSymbols('^GSPC',from='2008-01-01',to='2012-12-31')
data.model <- specifyModel(
  Next(100*Delt(Ad(GSPC))) ~ Delt(Ad(GSPC),k=1:10)+Delt(Vo(GSPC),k=1:3))
data <- as.data.frame(modelData(data.model))
colnames(data)[1] <- 'PercVarClose'
spExp <- performanceEstimation(
  PredTask(PercVarClose ~ .,data,'SP500_2012'),
  c(Workflow(wf='standardWF',wfID="standRF",
             learner='randomForest',
             learner.pars=list(ntree=500)),
    Workflow(wf='timeseriesWF',wfID="slideRF",
             learner='randomForest',
             learner.pars=list(ntree=500),
             type="slide",
             relearn.step=10)
   ),
  EstimationTask(
     metrics=c("mse","theil"),
     method=MonteCarlo(nReps=5,szTrain=0.5,szTest=0.25)
     )
)

## End(Not run)
```

`topPerformer` *Obtain the workflow that best performed in terms of a metric on a task*

Description

This function can be used to obtain the workflow (an object of class `Workflow`) that performed better in terms of a given metric on a certain task.

Usage

```
topPerformer(compRes,metric,task,max=FALSE,stat="avg")
```

Arguments

`compRes`	A `ComparisonResults` object with the results of your experimental comparison.
`metric`	A string with the name of a metric estimated in the comparison
`task`	A string with the name of a predictive task used in the comparison
`max`	A boolean (defaulting to `FALSE`) indicating the meaning of best performance for the selected metric. If this is `FALSE` it means that the goal is to minimize this metric, otherwise it means that the metric is to be maximized.
`stat`	The statistic to be used to obtain the ranks. The options are the statistics produced by the function `summary` applied to objects of class `ComparisonResults`, i.e. "avg", "std", "med", "iqr", "min", "max" or "invalid" (defaults to "avg").

Details

This is an utility function that can be used to obtain the workflow (an object of class Workflow) that achieved the best performance on a given predictive task in terms of a certain evaluation metric. The notion of *best performance* depends on the type of evaluation metric, thus the need for the max argument. Some evaluation statistics are to be maximized (e.g. accuracy), while others are to be minimized (e.g. mean squared error). For the former you should use max=TRUE, while the latter require max=FALSE (the default).

Value

The function returns an object of class Workflow .

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

performanceEstimation, topPerformers, rankWorkflows, metricsSummary

Examples

```
## Not run:
## Estimating several evaluation metrics on different variants of a
## regression tree and of a SVM, on  two data sets, using one repetition
## of  10-fold CV

data(swiss)
data(mtcars)
library(e1071)

## run the experimental comparison
results <- performanceEstimation(
               c(PredTask(Infant.Mortality ~ ., swiss),
                 PredTask(mpg ~ ., mtcars)),
               c(workflowVariants(learner='svm',
                                  learner.pars=list(cost=c(1,5),gamma=c(0.1,0.01))
                                 )
               ),
               EstimationTask(metrics=c("mse","mae"),method=CV(nReps=2,nFolds=5))
                                 )

## get the top performer workflow for a metric and task
topPerformer(results,"mse","swiss.Infant.Mortality")

## End(Not run)
```

topPerformers *Obtain the best scores from a performance estimation experiment*

Description

This function can be used to obtain the names of the workflows that obtained the best scores (the top performers) on an experimental comparison. This information will be shown for each of the evaluation metrics involved in the comparison and also for all predictive tasks that were used.

Usage

```
topPerformers(compRes,
          maxs=rep(FALSE,dim(compRes[[1]][[1]]@iterationsScores)[2]),
          stat="avg",digs=3)
```

Arguments

compRes	A ComparisonResults object with the results of your experimental comparison.
maxs	A vector of booleans with as many elements are there are metrics estimated in the experimental comparison. A TRUE value means the respective statistic is to be maximized, while a FALSE means minimization. Defaults to all FALSE values, i.e. all metrics are to be minimized.
stat	The statistic to be used to obtain the ranks. The options are the statistics produced by the function summary applied to objects of class ComparisonResults, i.e. "avg", "std", "med", "iqr", "min", "max" or "invalid" (defaults to "avg").
digs	The number of digits (defaults to 3) used in the scores column of the results.

Details

This is an utility function to check which were the top performers in a comparative experiment for each data set and each evaluation metric. The notion of *best performance* depends on the type of evaluation metric, thus the need for the second argument. Some evaluation statistics are to be maximized (e.g. accuracy), while others are to be minimized (e.g. mean squared error). If you have a mix of these types on your experiment then you can use the maxs parameter to inform the function of which are to be maximized and minimized.

Value

The function returns a list with named components. The components correspond to the predictive tasks used in the experimental comparison. For each component you get a data.frame, where the rows represent the statistics. For each statistic you get the name of the top performer (1st column of the data frame) and the respective score on that statistic (2nd column).

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

performanceEstimation, topPerformer, rankWorkflows, metricsSummary

Examples

```
## Not run:
## Estimating several evaluation metrics on different variants of a
## regression tree and of a SVM, on  two data sets, using one repetition
## of  10-fold CV

data(swiss)
data(mtcars)
library(e1071)

## run the experimental comparison
results <- performanceEstimation(
               c(PredTask(Infant.Mortality ~ ., swiss),
                 PredTask(mpg ~ ., mtcars)),
               c(workflowVariants(learner='svm',
                                  learner.pars=list(cost=c(1,5),gamma=c(0.1,0.01))
                                 )
               ),
               EstimationTask(metrics=c("mse","mae"),method=CV(nReps=2,nFolds=5))
                                 )
## get the top performers for each task and evaluation metric
topPerformers(results)

## End(Not run)
```

Workflow-class *Class "Workflow"*

Description

Objects of the class Workflow represent a solution to a predictive task that typically involve learning a prediction model using the given training data and then applying it to the provided test set. Still, a workflow function may carry out many more steps (e.g. special data pre-processing steps, or some form of post-processing of the predictions, etc.)

Objects from the Class

Objects can be created by calls of the form Workflow(...). This constructor function can generate 3 types of workflows: i) standard workflows; ii) time series standard workflows; and iii) user-defined workflows. The wf parameter of the constructor controls which type of workflow is created.

If the value of this parameter is "standardWF" or absent and no parameter type is included, then a workflow of type i) is created. If the value of wf is "timeseriesWF" or absent but the parameter type is set to either "slide" or "grow" then a workflow of type ii) is created. In all other cases a user-defined workflow is created which means that the function supplied in the parameter wf must exist and follow the input/output protocol of user-defined workflows. This protocol implies accepting a formula in the first argument, a training data frame in the second and a testing data frame in the third, with any other arguments being workflow-specific parameters. Moreover, the user-defined workflow must return a list object as result of its execution. See the Examples section for illustrations. The constructor can also be given a workflow ID in parameter wfID. Finally, the constructor also accepts a parameter deps with a valid value for the deps class slot.

Slots

name: An optional string containing an internal name of the workflow (a kind of ID)

func: A character string containing the name of the R function that implements the workflow.

pars: A named list containing the parameters and respective values to be used when calling the workflow (defaulting to the empty list).

deps: An optional named list with components "packages" and "scripts" each containing a vector of names of the required packages and scripts, respectively, for the workflow to be executable.

Methods

show `signature(object = "Workflow")`: method used to show the contents of a Workflow object.

summary `signature(object = "Workflow")`: method used to obtain a summary of a Workflow object.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

PredTask, EstimationTask, runWorkflow

Examples

```
showClass("Workflow")

## A simple standard workflow using a default svm
Workflow(learner="svm")

## Creating a standardWF
Workflow(learner="randomForest",learner.pars=list(ntree=420),wfID="rf420")
```

```
## Another one
Workflow(learner="svm",
         pre=c("centralImp","scale"),
         post="onlyPos",
         deps=list(packages=c("e1071"),scripts=c()))

## Another one
Workflow(learner="rpart",.fullOutput=TRUE)

## A user-defined workflow
myWF <- function(form,train,test,wL=0.5,...) {
    ml <- lm(form,train)
    mr <- rpart(form,train)
    pl <- predict(ml,test)
    pr <- predict(mr,test)
    ps <- wL*pl+(1-wL)*pr
    list(trues=responseValues(form,test),preds=ps)
}

wu <- Workflow(wf="myWF",wL=0.6)
```

`workflowNames` *The IDs of the workflows involved in an estimation experiment*

Description

This function can be used to get a vector with the IDs of the workflows that were used in a performance estimation experiment.

Usage

```
workflowNames(o)
```

Arguments

o	An object of class `ComparisonResults`

Value

A vector of strings (the names of the workflows)

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

ComparisonResults, performanceEstimation, taskNames, metricNames

Examples

```
## Not run:
## Estimating MSE for 3 variants of both
## regression trees and SVMs, on  two data sets, using one repetition
## of 10-fold CV
library(e1071)
library(DMwR)
data(swiss)
data(mtcars)

## running the estimation experiment
res <- performanceEstimation(
  c(PredTask(Infant.Mortality ~ .,swiss),PredTask(mpg ~ ., mtcars)),
  c(workflowVariants(learner="svm",
                     learner.pars=list(cost=c(1,10),gamma=c(0.01,0.5))),
    workflowVariants(learner="rpartXse",
                     learner.pars=list(se=c(0,0.5,1)))
  ),
  EstimationTask("mse")
  )

## the names of the metrics that were estimated in the above experiment
workflowNames(res)

## End(Not run)
```

---

workflowVariants *Generate (parameter) variants of a workflow*

Description

The main goal of this function is to facilitate the generation of different variants of a workflow. The idea is to be able to supply several possible values for a set of parameters of the workflow, and then have the function to return a set of Workflow objects, each consisting of one of the different possible combinations of the variants. This function finds its use in the context of performance estimation experiments, where we may actually be interested in comparing different parameter settings for a workflow.

Usage

```
workflowVariants(wf,...,varsRootName,as.is=NULL)
```

Arguments

wf	This is the string representing the name of the function of the base workflow from which variants should be generated. It can be ommited in the case of standard workflows (see Details section).
...	The function then accepts any number of named arguments, some of which may have vectors of values. These named arguments are supposed to be the names of parameters of the base workflow, and the parameters containing sets of values are the alternatives for the respective parameter that you want to consider in the variants generation (see examples below).
varsRootName	By default the names given to each variant will be formed by concatenating the base name of the workflow with the terminations: ".v1", ".v2", and so on. This parameter allows you to supply a different base name.
as.is	This is a vector of parameter names (defaults to NULL) that are not to be used as source for workflow variants. This is useful for workflows that have parameters that accept as "legal" values sets (e.g. a vector) and that we do not want the function workflowVariants to interprete as source values for generating different workflow variants but instead being use as they are given.

Value

The result of this function is a list of Workflow objects. Each of these objects represents one of the parameter variants of the workflow you have supplied.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2014) *An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R*. arXiv:1412.0436 [cs.MS] http://arxiv.org/abs/1412.0436

See Also

Workflow,performanceEstimation

Examples

```
## Not run:
## Generating several variants of the "svm" learner using different
## values of the parameter "cost", under the "umbrella" of a standard
## workflow (i.e. it assumes wf="standardWF")
library(e1071)
workflowVariants(learner="svm",cost=c(1,2,5,10))

## variants of a user defined workflow (assumes that function "userWF"
## is defined and "understands" parameters par1 and par2)
workflowVariants(wf="userWF",par1=c(0.1,0.4),par2=c(-10,10))
```

```
## Variants of a standard time series workflows (it assumes that it is a
## time series workflow because of the use of the "type" parameter,
## otherwise you could make it explicit by adding wf="timeseriesWF").
workflowVariants(learner="svm",type=c("slide","grow"),gamma=c(0.1,0.4))
## or equivalently
workflowVariants(wf="timeseriesWF",learner="svm",type=c("slide","grow"),gamma=c(0.1,0.4))

## allowing that one parameter is not considered for variants generation
workflowVariants(wf="userWF",par1=c(0.1,0.4),par2=c(-10,10),as.is="par1")

## nesting is also allowed!
workflowVariants(wf="userWF",
                 xpto=list(par1=c(0.1,0.4),d="k",par3=c(1,3)),
                 par2=c(-10,10),
                 as.is="par1")

## End(Not run)
```

Index