Package 'UBL'

July 13, 2017

Type Package

Title An Implementation of Re-Sampling Approaches to Utility-Based Learning for Both Classification and Regression Tasks

Description Provides a set of functions that can be used to obtain better predictive performance on cost-sensitive and cost/benefits tasks (for both regression and classification). This includes re-sampling approaches that modify the original data set biasing it towards the user preferences.

Version 0.0.6

Depends R(>= 3.0), methods, grDevices, graphics, stats, MBA, gstat, automap, sp, randomForest

Suggests MASS, rpart, testthat, DMwR, ggplot2, e1071

Date 2017-07-13

URL https://github.com/paobranco/UBL

BugReports https://github.com/paobranco/UBL/issues

License GPL (>= 2)

LazyLoad yes

LazyData yes

NeedsCompilation yes

Author Paula Branco [aut, cre], Rita Ribeiro [aut, ctb], Luis Torgo [aut, ctb]

Maintainer Paula Branco <paobranco@gmail.com>

Repository CRAN

Date/Publication 2017-07-13 13:01:35 UTC

R topics documented:

`UBL-package` *UBL: Utility-Based Learning*

Description

The package provides a diversity of pre-processing functions to deal with both classification (binary and multi-class) and regression problems that encompass non-uniform costs and/or benefits. These functions can be used to obtain a better predictive performance on this type of tasks. The package also includes two synthetic data sets for regression and classification.

Details

Name:	UBL
Type:	Package
Version:	0.0.5
Date:	2016-07-01
License:	GPL 2 GPL 3

The package in focused on utility-based learning, i.e., classification and regression problems with non-uniform benefits and/or costs. The main goal of the implemented functions is to improve the predictive performance of the models obtained. The package provides pre-processing approaches that change the original data set biasing it towards the user preferences.

All the methods avaliable are suitable for classification (binary and multiclass) and regression tasks. Moreover, several distance functions are also implemented which allows the use of the methods in data sets with categorical and/or numeric features.

We also provide two synthetic data sets for classification and regression.

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

Maintainer: Paula Branco

References

Branco, P., Ribeiro, R.P. and Torgo, L. (2016) *UBL: an R package for Utility-based Learning.* arXiv preprint arXiv:1604.08079.

Examples

```
## Not run:
library(UBL)
# an example with the synthetic classification data set provided with the package
data(ImbC)

plot(ImbC$X1, ImbC$X2, col = ImbC$Class, xlab = "X1", ylab = "X2")

summary(ImbC)
# randomly generate a 30-70% test and train partition
i.train <- sample(1:nrow(ImbC), as.integer(0.7*nrow(ImbC)))
trainD <- ImbC[i.train,]
testD <- ImbC[-i.train,]

model <- rpart(Class~., trainD)
preds <- predict(model, testD, type = "class")
table(preds, testD$Class)

# apply random over-sampling approach to balance the data set:

newTrain <- RandOverClassif(Class~., trainD)

newModel <- rpart(Class~., newTrain)
newPreds <- predict(newModel, testD, type = "class")
table(newPreds, testD$Class)


# an example with the synthetic regression data set provided with the package
data(ImbR)
```

```
library(ggplot2)
ggplot(ImbR, aes(x = X1, y = X2)) + geom_point(data = ImbR, aes(colour=Tgt)) +
      scale_color_gradient(low = "red", high="blue")

boxplot(ImbR$Tgt)
#relevance function automatically obtained
phiF.args <- phi.control(ImbR$Tgt, method = "extremes", extr.type = "high")
y.phi <- phi(sort(ImbR$Tgt),control.parms = phiF.args)

plot(sort(ImbR$Tgt), y.phi, type = "l", xlab = "Tgt variable", ylab = "relevance value")

# set the train and test data
i.train <- sample(1:nrow(ImbR), as.integer(0.7*nrow(ImbR)))
trainD <- ImbR[i.train,]
testD <- ImbR[-i.train,]

# train a model on the original train data
library(DMwR)
model <- rpartXse(Tgt~., trainD, se = 0)

preds <- predict(model, testD)

plot(testD$Tgt, preds, xlim = c(0,55), ylim = c(0,55))
abline(a = 0, b = 1)

# obtain a new train using random under-sampling strategy
newTrain <- RandUnderRegress(Tgt~., trainD)
newModel <- rpartXse(Tgt~., newTrain, se = 0)
newPreds <- predict(newModel, testD)

# plot the predictions for the model obtained with
# the original and the modified train data
plot(testD$Tgt, preds, xlim = c(0,55), ylim = c(0,55)) #black for original train
abline(a = 0, b = 1, lty=2, col="grey")
points(testD$Tgt, newPreds, col="blue", pch=2) #blue for changed train
abline(h=30, lty=2, col="grey")
abline(v=30, lty=2, col="grey")

## End(Not run)
```

`AdasynClassif` *ADASYN algorithm for unbalanced classification problems, both binary and multi-class.*

Description

This function handles unbalanced classification problems using the ADASYN algorithm. This algorithm generates synthetic cases using a SMOTE-like approache. However, the examples of the class(es) where over-sampling is applied are weighted according to their level of difficulty in learning. This means that more synthetic data is generated for cases which are hrder to learn compared

to the examples of the same class that are easier to learn. This implementation provides a strategy suitable for both binary and multi-class problems.

Usage

```
AdasynClassif(form, dat, baseClass = NULL, beta = 1, dth = 0.95,
                          k = 5, dist = "Euclidean", p = 2)
```

Arguments

`form`	A formula describing the prediction problem
`dat`	A data frame containing the original (unbalanced) data set
`baseClass`	Character specifying the reference class, i.e., the class from which all other classes will be compared to. This can be selected by the user or estimated from the classes distribution. If not defined (the default) the majority class is selected.
`beta`	Either a numeric value indicating the desired balance level after synthetic examples generation, or a named list specifying the selected classes beta value. A beta value of 1 (the default) corresponds to full balancing the classes. See examples.
`dth`	
`k`	A number indicating the number of nearest neighbors that are used to generate the new examples of the minority class(es).
`dist`	A character string indicating which distance metric to use when determining the k nearest neighbors. See the details. Defaults to "Euclidean".
`p`	A number indicating the value of p if the "p-norm" distance is chosen. Only necessary to define if a "p-norm" is chosen in the `dist` argument. See details.

Details

`dist` **parameter:** The parameter `dist` allows the user to define the distance metric to be used in the neighbors computation. Although the default is the Euclidean distance, other metrics are available. This allows the computation of distances in data sets with, for instance, both nominal and numeric features. The options available for the distance functions are as follows:

- for data with only numeric features: "Manhattan", "Euclidean", "Canberra", "Chebyshev", "p-norm";

- for data with only nominal features: "Overlap";

- for dealing with both nominal and numeric features: "HEOM", "HVDM".

When the "p-norm" is selected for the `dist` parameter, it is also necessary to define the value of parameter `p`. The value of parameter `p` sets which "p-norm" will be used. For instance, if `p` is set to 1, the "1-norm" (or Manhattan distance) is used, and if `p` is set to 2, the "2-norm" (or Euclidean distance) is applied. For more details regarding the distance functions implemented in UBL package please see the package vignettes.

Value

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

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

References

He, H., Bai, Y., Garcia, E.A. and Li, S., 2008, June. *ADASYN: Adaptive synthetic sampling approach for imbalanced learning.* In 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence) (pp. 1322-1328). IEEE.

See Also

SmoteClassif, RandOverClassif, ImpSampClassif

Examples

```
# Example with an imbalanced multi-class problem
 data(iris)
 dat <- iris[-c(45:75), c(1, 2, 5)]
# checking the class distribution of this artificial data set
 table(dat$Species)
 newdata <- AdasynClassif(Species~., dat, beta=1)
 table(newdata$Species)
 beta <- list("setosa"=1, "versicolor"=0.5)
 newdata <- AdasynClassif(Species~., dat, baseClass="virginica", beta=beta)
 table(newdata$Species)

## Checking visually the created data
par(mfrow = c(1, 2))
plot(dat[, 1], dat[, 2], pch = 19 + as.integer(dat[, 3]),
     col = as.integer(dat[,3]), main = "Original Data",
     xlim=range(newdata[,1]), ylim=range(newdata[,2]))
plot(newdata[, 1], newdata[, 2], pch = 19 + as.integer(newdata[, 3]),
     col = as.integer(newdata[,3]), main = "New Data",
     xlim=range(newdata[,1]), ylim=range(newdata[,2]))

# A binary example
library(MASS)
data(cats)
table(cats$Sex)
Ada1cats <- AdasynClassif(Sex~., cats)
table(Ada1cats$Sex)
Ada2cats <- AdasynClassif(Sex~., cats, beta=5)
table(Ada2cats$Sex)
```

`CNNClassif` *Condensed Nearest Neighbors strategy for multiclass imbalanced problems*

Description

This function applies the Condensed Nearest Neighbors (CNN) strategy for imbalanced multiclass problems. It constructs a subset of examples which are able to correctly classify the original data set using a one nearest neighbor rule.

Usage

```
CNNClassif(form, dat, dist = "Euclidean", p = 2, Cl = "smaller")
```

Arguments

`form`	A formula describing the prediction problem.
`dat`	A data frame containing the original imbalanced data set.
`dist`	A character string indicating which distance metric to use when determining the k nearest neighbors. See the details. Defaults to "Euclidean".
`p`	A number indicating the value of p if the "p-norm" distance is chosen. Only necessary to define if a "p-norm" is chosen in the `dist` argument. See details.
`Cl`	A character vector indicating which are the most important classes. Defaults to "smaller" which means that the smaller classes are automatically determined. In this case, all the smaller classes are those with a frequency below #examples/#classes. With the selection of option "smaller" those classes are the ones considered important for the user.

Details

`dist` **parameter:** The parameter `dist` allows the user to define the distance metric to be used in the neighbors computation. Although the default is the Euclidean distance, other metrics are available. This allows the computation of distances in data sets with, for instance, both nominal and numeric features. The options available for the distance functions are as follows:

- for data with only numeric features: "Manhattan", "Euclidean", "Canberra", "Chebyshev", "p-norm";

- for data with only nominal features: "Overlap";

- for dealing with both nominal and numeric features: "HEOM", "HVDM".

When the "p-norm" is selected for the `dist` parameter, it is also necessary to define the value of parameter `p`. The value of parameter `p` sets which "p-norm" will be used. For instance, if `p` is set to 1, the "1-norm" (or Manhattan distance) is used, and if `p` is set to 2, the "2-norm" (or Euclidean distance) is applied. For more details regarding the distance functions implemented in UBL package please see the package vignettes.

CNN algorithm: This function applies the Condensed Nearest Neighbors (CNN) strategy for dealing with imbalanced multiclass problems. The classes selected in `Cl` are considered the most important ones and all the others are under-sampled. The CNN under-sampling strategy starts with a set composed by all the examples from the important classes and one randomly selected example from the other classes. Then, examples from the other classes are added to the set forming a subset of examples which correctly classifies the original data set using a one nearest neighbor rule.

Value

The function returns a list with a data frame with the new data set resulting from the application of the CNN strategy, a character vector with the important classes, and another character vector with the unimportant classes.

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Hart, P. E. (1968). *The condensed nearest neighbor rule* IEEE Transactions on Information Theory, 14, 515-516

See Also

`OSSClassif`, `TomekClassif`

Examples

```
library(DMwR)
data(algae)
clean.algae <- algae[complete.cases(algae), ]
myCNN <- CNNClassif(season~., clean.algae,
                    Cl = c("summer", "spring", "winter"),
                    dist = "HEOM")
CNN1 <- CNNClassif(season~., clean.algae, Cl = "smaller", dist = "HEOM")
CNN2<- CNNClassif(season~., clean.algae, Cl = "summer",dist = "HVDM")
summary(myCNN[[1]]$season)
summary(CNN1[[1]]$season)
summary(CNN2[[1]]$season)


library(MASS)
data(cats)
CNN.catsF <- CNNClassif(Sex~., cats, Cl = "F")
CNN.cats <- CNNClassif(Sex~., cats, Cl = "smaller")
```

`distances` *Distance matrix between all data set examples according to a selected distance metric.*

Description

This function computes the distances between all examples in a data set using a selected distance metric. The metrics available are suitable for data sets with numeric and/or nominal features and include, among others: Euclidean, Manhattan, HEOM and HVDM.

Usage

```
distances(tgt, dat, dist, p=2)
```

Arguments

`tgt`	The column of the problem target variable.
`dat`	A data frame containing the problem data.
`dist`	A character string specifying the distance function to use in the nearest neighbours evaluation.
`p`	An optional parameter that is only required if the distance function selected in parameter `dist` is "p-norm".

Details

Several distance function are implemented in UBL package. The goal of having such a diversity of distance functions is to provide the users more flexibility regarding the distance used and also to provide distance fucntions that are able to deal with nominal and numeric features. The options available for the distance functions are as follows:

data with only numeric features: "Manhattan", "Euclidean", "Canberra", "Chebyshev", "p-norm";

data with only nominal features: "Overlap";

data with both nominal and numeric features: "HEOM", "HVDM".

When the "p-norm" is selected for the `dist` parameter, it is also necessary to define the value of parameter `p`. The value of parameter `p` sets which "p-norm" will be used. For instance, if `p` is set to 1, the "1-norm" (or Manhattan distance) is used, and if `p` is set to 2, the "2-norm" (or Euclidean distance) is applied. For more details regarding the distance functions implemented in UBL package please see the package vignettes.

Value

The function returns a matrix with the distances computed between each pair of examples in the data set.

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Wilson, D.R. and Martinez, T.R. (1997). *Improved heterogeneous distance functions.* Journal of artificial intelligence research, pp.1-34.

See Also

neighbours

Examples

```
## Not run:
data(ImbC)
# determine the distances between each example in ImbC data set
# using the "HVDM" distance function.
dist1 <- distances(3, ImbC, "HVDM")

# now using the "HEOM" distance function
dist2 <- distances(3, ImbC, "HEOM")

# check the differences
head(dist1)
head(dist2)

## End(Not run)
```

ENNClassif *Edited Nearest Neighbor for multiclass imbalanced problems*

Description

This function handles imbalanced classification problems using the Edited Nearest Neighbor (ENN) algorithm. It removes examples whose class label differs from the class of at least half of its k nearest neighbors. All the existing classes can be under-sampled with this technique. Alternatively a subset of classes to under-sample can be provided by the user.

Usage

```
ENNClassif(form, dat, k = 3, dist = "Euclidean", p = 2, Cl = "all")
```

Arguments

form	A formula describing the prediction problem.
dat	A data frame containing the original (imbalanced) data set.
k	A number indicating the number of nearest neighbors to use.
dist	A character string indicating which distance metric to use when determining the k nearest neighbors. See the details. Defaults to "Euclidean".
p	A number indicating the value of p if the "p-norm" distance is chosen. Only necessary to define if a "p-norm" is chosen in the dist argument. See details.
Cl	A character vector indicating which classes should be under-sampled. Defaults to "all" meaning that all classes are candidates for having examples removed. The user may define a subset of the existing classes in which this technique will be applied.

Details

dist **parameter:** The parameter dist allows the user to define the distance metric to be used in the neighbors computation. Although the default is the Euclidean distance, other metrics are available. This allows the computation of distances in data sets with, for instance, both nominal and numeric features. The options available for the distance functions are as follows:

- for data with only numeric features: "Manhattan", "Euclidean", "Canberra", "Chebyshev", "p-norm";

- for data with only nominal features: "Overlap";

- for dealing with both nominal and numeric features: "HEOM", "HVDM".

When the "p-norm" is selected for the dist parameter, it is also necessary to define the value of parameter p. The value of parameter p sets which "p-norm" will be used. For instance, if p is set to 1, the "1-norm" (or Manhattan distance) is used, and if p is set to 2, the "2-norm" (or Euclidean distance) is applied. For more details regarding the distance functions implemented in UBL package please see the package vignettes.

ENN algorithm: The ENN algorithm uses a cleaning method to perform under-sampling. For each example with class label in Cl the k nearest neighbors are computed using a selected distance metric. The example is removed from the data set if it is misclassified by at least half of it's k nearest neighbors. Usually this algorithm uses k=3.

Value

The function returns a list containing a data frame with the new data set resulting from the application of the ENN algorithm, and the indexes of the examples removed.

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

References

D. Wilson. (1972). *Asymptotic properties of nearest neighbor rules using edited data*. Systems, Man and Cybernetics, IEEE Transactions on, 408-421.

See Also

NCLClassif

Examples

```
# generate an small imbalanced data set
  ir<- iris[-c(95:130), ]
# use ENN technique with different metrics, number of neighbours and classes
  ir1norm <- ENNClassif(Species~., ir, k = 5, dist = "p-norm",
                        p = 1, Cl = "all")
  irEucl <- ENNClassif(Species~., ir) # defaults to Euclidean distance
  irCheby <- ENNClassif(Species~., ir, k = 7, dist = "Chebyshev",
                       Cl = c("virginica", "setosa"))
  irHVDM <- ENNClassif(Species~., ir, k = 3, dist = "HVDM")
# checking the impact
  summary(ir$Species)
  summary(ir1norm[[1]]$Species)
  summary(irEucl[[1]]$Species)
  summary(irCheby[[1]]$Species)
  summary(irHVDM[[1]]$Species)
# check the removed indexes of the ir1norm data set
  ir1norm[[2]]
```

EvalClassifMetrics *Utility metrics for assessing the performance of utility-based classification tasks.*

Description

This function allows to evaluate utility-based metrics in classification problems which have defined a cost, benefit, or utility matrix.

Usage

```
EvalClassifMetrics(trues, preds, mtr, type = "util", metrics = NULL, thr=0.5, beta = 1)
```

Arguments

trues	A vector with the true target variable values of the problem.
preds	A vector with the prediction values obtained for the vector of trues.
mtr	A matrix that can be either a cost, a benefit or a utility matrix. The matrix must be always provided with the true class in the rows and the predicted class in the columns.
type	A character specifying the type of matrix provided. Can be set to "cost", "benefit" or "utility" (the default).
metrics	A character vector with the metrics names to be evaluated. If not specified (the default), all the metrics avaliable for the type of matrix provided are evaluated.

thr — A numeric value between 0 and 1 setting a threshold on the relevance values for determining which are the important classes to consider. This threshold is only necessary for the following metrics: precPhi, recPhi and FPhi. Moreover, these metrics are only available for problems based on utility matrices. Defaults to 0.5.

beta — The numeric value of the beta parameter for F-score.

Value

The function returns a named list with the evaluated metrics results.

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Ribeiro, R., 2011. Utility-based regression (Doctoral dissertation, PhD thesis, Dep. Computer Science, Faculty of Sciences - University of Porto).

Branco, P., 2014. Re-sampling Approaches for Regression Tasks under Imbalanced Domains (Msc thesis, Dep. Computer Science, Faculty of Sciences - University of Porto).

See Also

phi.control

Examples

```
# the synthetic data set provided with UBL package for classification
data(ImbC)
sp <- sample(1:nrow(ImbC), round(0.7*nrow(ImbC)))
train <- ImbC[sp, ]
test <- ImbC[-sp,]

# example with a utility matrix
# define a utility matrix (true class in rows and pred class in columns)
 matU <- matrix(c(0.2, -0.5, -0.3, -1, 1, -0.9, -0.9, -0.8, 0.9), byrow=TRUE, ncol=3)
# determine optimal preds (predictions that maximize utility)
library(e1071) # for the naiveBayes classifier
 resUtil <- UtilOptimClassif(Class~., train, test, mtr = matU, type="util",
                       learner = "naiveBayes",
                       predictor.pars = list(type="raw", threshold = 0.01))

# learning a model without maximizing utility
 model <- naiveBayes(Class~., train)
 resNormal <- predict(model, test, type="class", threshold = 0.01)
#Check the difference in the total utility of the results
 EvalClassifMetrics(test$Class, resNormal, mtr=matU, type= "util")
 EvalClassifMetrics(test$Class, resUtil, mtr=matU, type= "util")
```

```
# example with a classification task that has a cost matrix associated
# define a cost matrix (true class in rows and pred class in columns)
 matC <- matrix(c(0, 0.5, 0.3, 1, 0, 0.9, 0.9, 0.8, 0), byrow=TRUE, ncol=3)
 resUtil <- UtilOptimClassif(Class~., train, test, mtr = matC, type="cost",
                             learner = "naiveBayes",
                             predictor.pars = list(type="raw", threshold = 0.01))

 # learning a model without maximizing utility
 model <- naiveBayes(Class~., train)
 resNormal <- predict(model, test, type="class")
 #Check the difference in the total utility of the results
 EvalClassifMetrics(test$Class, resNormal, mtr=matC, type= "cost")
 EvalClassifMetrics(test$Class, resUtil, mtr=matC, type= "cost")

#example with a benefit matrix
# define a benefit matrix (true class in rows and pred class in columns)
 matB <- matrix(c(0.2, 0, 0, 0, 1, 0, 0, 0, 0.9), byrow=TRUE, ncol=3)

 resUtil <- UtilOptimClassif(Class~., train, test, mtr = matB, type="ben",
                             learner = "naiveBayes",
                             predictor.pars = list(type="raw", threshold = 0.01))

# learning a model without maximizing utility
 model <- naiveBayes(Class~., train)
 resNormal <- predict(model, test, type="class", threshold = 0.01)
# Check the difference in the total utility of the results
 EvalClassifMetrics(test$Class, resNormal, mtr=matB, type= "ben")
 EvalClassifMetrics(test$Class, resUtil, mtr=matB, type= "ben")

 table(test$Class,resNormal)
 table(test$Class,resUtil)
```

`EvalRegressMetrics` *Utility metrics for assessing the performance of utility-based regression tasks.*

Description

This function allows to evaluate utility-based metrics in regression problems which have defined a cost, benefit, or utility surface.

Usage

```
EvalRegressMetrics(trues, preds, util.vals, type = "util",
metrics = NULL, thr = 0.5, control.parms = NULL,
beta = 1, maxC = NULL, maxB = NULL)
```

Arguments

trues	A vector with the true target variable values of the problem.
preds	A vector with the prediction values obtained for the vector of trues.
util.vals	Either the cost, benefit or utility values corresponding to the provided points (trues, preds).
type	A character specifying the type of surface under consideration. Can be set to "cost", "benefit" or "utility" (the default).
metrics	A character vector with the metrics names to be evaluated. If not specified (the default), all the metrics avaliable for the type of surface provided are evaluated.
thr	A numeric value between 0 and 1 setting a threshold on the relevance values for determining which are the important cases to consider. This threshold is only necessary for the following metrics: precPhi, recPhi and FPhi. Moreover, these metrics are only available for problems based on utility surfaces. Defaults to 0.5.
control.parms	the control.parms of the relevance function phi. Can be obtained through function phi.control. These are only necessary for evaluating the following utility metrics: recPhi, precPhi and FPhi.
beta	The numeric value of the beta parameter for F-score.
maxC	the maximum cost achievable in the cost surface. Parameter only required when the problem depends on a cost surface.
maxB	the maximum Benefit achievable in the benefit surface. Parameter only required when the problem depends on a benefit surface.

Value

The function returns a named list with the evaluated metrics results.

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Ribeiro, R., 2011. Utility-based regression (Doctoral dissertation, PhD thesis, Dep. Computer Science, Faculty of Sciences - University of Porto).

Branco, P., 2014. Re-sampling Approaches for Regression Tasks under Imbalanced Domains (Msc thesis, Dep. Computer Science, Faculty of Sciences - University of Porto).

See Also

phi.control

Examples

```
## Not run:
#Example using a utility surface interpolated and observing the performance of
# two models: i) a model obtained with a strategy designed for maximizing
# predictions utility and a model obtained through a standard random Forest.

data(Boston, package = "MASS")

tgt <- which(colnames(Boston) == "medv")
sp <- sample(1:nrow(Boston), as.integer(0.7*nrow(Boston)))
train <- Boston[sp,]
test <- Boston[-sp,]

control.parms <- phi.control(Boston[,tgt], method="extremes", extr.type="both")
# the boundaries of the domain considered
minds <- min(train[,tgt])
maxds <- max(train[,tgt])

# build m.pts to include at least (minds, maxds) and (maxds, minds) points
# m.pts must only contain points in [minds, maxds] range.
m.pts <- matrix(c(minds, maxds, -1, maxds, minds, -1),
                byrow=TRUE, ncol=3)

pred.res <- UtilOptimRegress(medv~., train, test, type = "util", strat = "interpol",
                             strat.parms=list(method = "bilinear"),
                             control.parms = control.parms,
                             m.pts = m.pts, minds = minds, maxds = maxds)

# assess the performance
eval.util <- EvalRegressMetrics(test$medv, pred.res$optim, pred.res$utilRes,
                                thr = 0.8, control.parms = control.parms)

# now train a normal model
model <- randomForest(medv~.,train)
normal.preds <- predict(model, test)

#obtain the utility of this model predictions
NormalUtil <- UtilInterpol(test$medv, normal.preds, type = "util",
                           control.parms = control.parms,
                           minds, maxds, m.pts, method = "bilinear")

#check the performance
eval.normal <- EvalRegressMetrics(test$medv, normal.preds, NormalUtil,
                                  thr=0.8, control.parms = control.parms)

# 3 check visually the utility surface and the predictions of both models
UtilInterpol(NULL,NULL, type = "util", control.parms = control.parms,
                           minds, maxds, m.pts, method = "bilinear",
                           visual=TRUE, full.output = TRUE)
points(test$medv, normal.preds) # standard model predition points
points(test$medv, pred.res$optim, col="blue") # model with optimized predictions
```

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

`GaussNoiseClassif` *Introduction of Gaussian Noise for the generation of synthetic examples to handle imbalanced multiclass problems.*

Description

This strategy performs both over-sampling and under-sampling. The under-sampling is randomly performed on the examples of the classes defined by the user through the `C.perc` parameter. Regarding the over-sampling method, this is based on the generation of new synthetic examples with the introduction of a small perturbation on existing examples through Gaussian noise. A new example from a minority class is obtained by perturbing each feature a percentage of its standard deviation (evaluated on the minority class examples). For nominal features, the new example randomly selects a label according to the frequency of examples belonging to the minority class. The `C.perc` parameter is also used to express which percentage of over-sampling should be applied and to which classes.

Usage

```
GaussNoiseClassif(form, dat, C.perc = "balance", pert = 0.1, repl = FALSE)
```

Arguments

`form`	A formula describing the prediction problem
`dat`	A data frame containing the original (unbalanced) data set
`C.perc`	A named list containing the percentage(s) of under- or/and over-sampling to apply to each class. The over-sampling percentage is a number above 1 while the under-sampling percentage should be a number below 1. If the number 1 is provided for a given class then that class remains unchanged. Alternatively it may be "balance" (the default) or "extreme", cases where the sampling percentages are automatically estimated either to balance the examples between the minority and majority classes or to invert the distribution of examples across the existing classes transforming the majority classes into minority and vice-versa.
`pert`	A number indicating the level of perturbation to introduce when generating synthetic examples. Assuming as center the base example, this parameter defines the radius (based on the standard deviation) where the new example is generated.
`repl`	A boolean value controlling the possibility of having repetition of examples when performing under-sampling by selecting among the majority class(es) examples.

Value

The function returns a data frame with the new data set resulting from the application of random under-sampling and over-sampling through the generation of synthetic examples using Gaussian noise.

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Sauchi Stephen Lee. (1999) *Regularization in skewed binary classification.* Computational Statistics Vol.14, Issue 2, 277-292.

Sauchi Stephen Lee. (2000) *Noisy replication in skewed binary classification.* Computaional stistics and data analysis Vol.34, Issue 2, 165-191.

See Also

SmoteClassif

Examples

```
library(DMwR)
data(algae)
clean.algae <- algae[complete.cases(algae), ]
# autumn and summer are the most important classes and winter
# is the least important
C.perc = list(autumn = 3, summer = 1.5, winter = 0.2)
gn <- GaussNoiseClassif(season~., clean.algae, C.perc)
table(algae$season)
table(gn$season)

# another example
data(iris)
dat <- iris[, c(1, 2, 5)]
dat$Species <- factor(ifelse(dat$Species == "setosa", "rare", "common"))
## checking the class distribution of this artificial data set
table(dat$Species)
## now using gaussian noise to create a more "balanced problem"
new.gn <- GaussNoiseClassif(Species ~ ., dat)
table(new.gn$Species)
## Checking visually the created data
 par(mfrow = c(1, 2))
 plot(dat[, 1], dat[, 2], pch = as.integer(dat[, 3]),
      col = as.integer(dat[, 3]), main = "Original Data")
 plot(new.gn[, 1], new.gn[, 2], pch = as.integer(new.gn[, 3]),
      col = as.integer(new.gn[, 3]), main = "Data with Gaussian Noise")
```

GaussNoiseRegress *Introduction of Gaussian Noise for the generation of synthetic examples to handle imbalanced regression problems*

Description

This strategy performs both over-sampling and under-sampling. The under-sampling is randomly performed on the examples below the relevance threshold defined by the user. Regarding the over-sampling method, this is based on the generation of new synthetic examples with the introduction of a small perturbation on existing examples through Gaussian noise. A new example from a rare "class"" is obtained by perturbing all the features and the target variable a percentage of its standard deviation (evaluated on the rare examples). The value of nominal features of the new example is randomly selected according to the frequency of the values existing in the rare cases of the bump in consideration.

Usage

```
GaussNoiseRegress(form, dat, rel = "auto", thr.rel = 0.5, C.perc = "balance",
                  pert = 0.1, repl = FALSE)
```

Arguments

`form`	A formula describing the prediction problem
`dat`	A data frame containing the original (unbalanced) data set
`rel`	The relevance function which can be automatically ("auto") determined (the default) or may be provided by the user through a matrix with interpolating points.
`thr.rel`	A number indicating the relevance threshold above which a case is considered as belonging to the rare "class".
`C.perc`	A list containing the percentage(s) of under- or/and over-sampling to apply to each "class" (bump) obtained with the threshold. The `C.perc` values should be provided in ascending order of target variable values. The over-sampling percentage(s) should be numbers above 1 and represent the increase that is applied to the examples of the bump. The under-sampling percentage(s) should be numbers below 1 and represent the decrease applied to the cases in the corresponding bump. If the value of 1 is provided for a given bump, then the examples in that bump will remain unchanged. Alternatively it may be "balance" (the default) or "extreme", cases where the sampling percentages are automatically estimated.
`pert`	A number indicating the level of perturbation to introduce when generating synthetic examples. Assuming as center the base example, this parameter defines the radius (based on the standard deviation) where the new example is generated.
`repl`	A boolean value controlling the possibility of having repetition of examples when performing under-sampling by selecting among the "normal" examples.

Value

The function returns a data frame with the new data set resulting from the application of random under-sampling and over-sampling through the generation of synthetic examples using Gaussian noise.

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Sauchi Stephen Lee. (1999) *Regularization in skewed binary classification.* Computational Statistics Vol.14, Issue 2, 277-292.

Sauchi Stephen Lee. (2000) *Noisy replication in skewed binary classification.* Computaional stistics and data analysis Vol.34, Issue 2, 165-191.

See Also

SmoteRegress

Examples

```
library(DMwR)
data(algae)
clean.algae <- algae[complete.cases(algae), ]
C.perc = list(0.5, 3)
mygn.alg <- GaussNoiseRegress(a7~., clean.algae, C.perc = C.perc)
gnB.alg <- GaussNoiseRegress(a7~., clean.algae, C.perc = "balance",
                             pert = 0.1)
gnE.alg <- GaussNoiseRegress(a7~., clean.algae, C.perc = "extreme")

plot(density(clean.algae$a7))
lines(density(gnE.alg$a7), col = 2)
lines(density(gnB.alg$a7), col = 3)
lines(density(mygn.alg$a7), col = 4)



ir <- iris[-c(95:130), ]
mygn1.iris <- GaussNoiseRegress(Sepal.Width~., ir, C.perc = list(0.5, 2.5))
mygn2.iris <- GaussNoiseRegress(Sepal.Width~., ir, C.perc = list(0.2, 4),
                                thr.rel = 0.8)
gnB.iris <- GaussNoiseRegress(Sepal.Width~., ir, C.perc = "balance")
gnE.iris <- GaussNoiseRegress(Sepal.Width~., ir, C.perc = "extreme")

# defining a relevance function
rel <- matrix(0, ncol = 3, nrow = 0)
rel <- rbind(rel, c(2, 1, 0))
rel <- rbind(rel, c(3, 0, 0))
rel <- rbind(rel, c(4, 1, 0))

gn.rel <- GaussNoiseRegress(Sepal.Width~., ir, rel = rel,
                            C.perc = list(5, 0.2, 5))
plot(density(ir$Sepal.Width), ylim = c(0,1))
lines(density(gnB.iris$Sepal.Width), col = 3)
lines(density(gnE.iris$Sepal.Width, bw = 0.3), col = 4)
# check the impact of a different relevance threshold
lines(density(gn.rel$Sepal.Width), col = 2)
```

ImbC *Synthetic Imbalanced Data Set for a Multi-class Task*

Description

Synthetic imbalanced data set for a multi-class task. The data set has a numeric feature ("X1"), a nominal feature ("X2") and a target class named "Class". The three classes of the problem ("normal", "rare1" and "rare2") are assigned according to the rules described below. These rules depend of the two features ("X1" and "X2").

Usage

```
data(ImbC)
```

Format

The data set has one continuous feature (`X1`) and one nominal feature (`X2`). The target class (denoted as `Class`) has three possible values ("normal" , "rare1" and "rare2"). Classes "rare1" and "rare2" are the minority classes. Examples of class "rare1" occur in 1% of the data while those of class "rare2" occur in 13.1% of the data. The remaining class, "normal", is the majority class and occurs in about 85.9% of the data. Data set ImbC has 1000 examples distributed in classes "rare1", "rare2" and "normal" with 10, 131 and 859 examples respectively.

ImbC data has been simulated as follows:

- `X1`$\sim \mathbf{N}(0, 4)$
- `X2` labels "cat", "fish" and "dog" where randomly distributed with the restriction of having a frequency of 30%, 30% and 40% respectively.
- To obtain the target variable `Class`, we have define the following sets:
  - $S_1 = \{(X1, X2) : X1 > 9 \wedge (X2 \in \{"cat", "dog"\})\}$
  - $S_2 = \{(X1, X2) : X1 > 7 \wedge X2 = "fish"\}$
  - $S_3 = \{(X1, X2) : -1 < X1 < 0.5\}$
  - $S_4 = \{(X1, X2) : X1 < -7 \wedge X2 = "fish"\}$
- The following conditions define the target variable distribution of the ImbC synthetic data set:
  - Assign class label "rare1" to: a random sample of 90% of set S_1 and a random sample of 40% of set S_2
  - Assign class label "rare2" to: a random sample of 80% of set S_3 and a random sample of 70% of set S_4
  - Assign class label "normal" to the remaing examples.

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

Examples

```
require(ggplot2)
data(ImbC)
summary(ImbC)
ggplot(data=ImbC, aes(x=X2, y=X1, color=Class))+geom_jitter()
```

ImbR *Synthetic Regression Data Set*

Description

Simulated data set for imbalanced domain on regression. The rare cases corresponden to the higher extreme values and are described by a circle with white noise. The normal cases have a normal distribution with the same center of the circunference with elliptical contours.

Usage

```
data(ImbR)
```

Format

The data set has 2 continuous features (`X1` and `X2`) and a continuous target variable (denoted as `Tgt`). The rare examples, i.e, cases with higher values of the target variable occur in 5% of the data. Data set ImbR has 1000 examples.

ImbR data has been simulated as follows:

- lower `Tgt` values: `(X1, X2)`$\sim \mathbf{N}_2\left(\mathbf{10}_2, \mathbf{2.5}_2\right)$ and `Tgt`$\sim \mathbf{\Gamma}\left(0.5, 1\right) + 10$
- higher `Tgt` values: `(X1, X2)`$\sim (\rho * cos(\theta) + 10, \rho * sin(\theta) + 10)$, where $\rho \sim \mathbf{9}_2 + \mathbf{N}_2\left(\mathbf{0}_2, \mathbf{I}_2\right)$ and $\theta \sim \mathbf{U}_2\left(\mathbf{0}_2, 2\pi\mathbf{I}_2\right)$ `Tgt`$\sim \mathbf{\Gamma}\left(1, 1\right) + 20$

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

Examples

```
data(ImbR)
summary(ImbR)

boxplot(ImbR$Tgt)
```

ImpSampClassif *Importance Sampling algorithm for imbalanced classification problems*

Description

This function handles imbalanced classification problems using the importance/relevance provided to re-sample the data set. The relevance is used to introduce replicas of the most important examples and to remove the least important examples. This function combines random over-sampling with random under-sampling which are applied in the problem classes according to the corresponding relevance.

Usage

```
ImpSampClassif(form, dat, C.perc = "balance")
```

Arguments

form	A formula describing the prediction problem
dat	A data frame containing the original (unbalanced) data set
C.perc	A list containing the percentage(s) of random under- or over-sampling to apply to each class. The over-sampling percentage is a number above 1 while the under-sampling percentage should be a number below 1. If the number 1 is provided for a given class then that class remains unchanged. Alternatively it may be "balance" (the default) or "extreme", cases where the sampling percentages are automatically estimated.

Value

The function returns a data frame with the new data set resulting from the application of the importance sampling strategy.

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

See Also

RandUnderClassif, RandOverClassif

Examples

```
data(iris)
# generating an artificially imbalanced data set
ir <- iris[-c(51:70,111:150), ]
IS.ext <-ImpSampClassif(Species~., ir, C.perc = "extreme")
IS.bal <-ImpSampClassif(Species~., ir, C.perc = "balance")
```

```
myIS <-ImpSampClassif(Species~., ir, C.perc = list(setosa = 0.2,
                                                   versicolor = 2,
                                                   virginica = 6))
# check the results
table(ir$Species)
table(IS.ext$Species)
table(IS.bal$Species)
table(myIS$Species)
```

`ImpSampRegress` *Importance Sampling algorithm for imbalanced regression problems*

Description

This function handles imbalanced regression problems using the relevance function provided to re-sample the data set. The relevance function is used to introduce replicas of the most important examples and to remove the least important examples.

Usage

```
ImpSampRegress(form, dat, rel = "auto", thr.rel = NA,
               C.perc = "balance", O = 0.5, U = 0.5)
```

Arguments

`form`	A formula describing the prediction problem
`dat`	A data frame containing the original (unbalanced) data set
`rel`	The relevance function which can be automatically ("auto") determined (the default) or may be provided by the user through a matrix with interpolating points.
`thr.rel`	The default is NA which means that no threshold is used when performing the over/under-sampling. In this case, the over-sampling is performed by assigning a higher probability for selecting an example to the examples with higher relevance. On the other hand, the under-sampling is performed by removing the examples with less relevance. The user may chose a number between 0 and 1 indicating the relevance threshold above which a case is considered as belonging to the rare "class".
`C.perc`	A list containing the percentage(s) of under- or/and over-sampling to apply to each "class" obtained with the threshold. This parameter is only used when a relevance threshold (thr.rel) is set. Otherwise it is ignored. The `C.perc` values should be provided in ascending order of target variable values. The over-sampling percentage(s) must be numbers higher than 1 and represent the increase applied to the examples of the bump. The under-sampling percentage(s) should be numbers below 1 and represent the decrease applyed in the corresponding bump. If the value of 1 is provided for a given bump, then the examples in that bump will remain unchanged. Alternatively, this parameter may be set to "balance" (the default) or "extreme", cases where the sampling percentages are automatically estimated.

O — A number expressing the importance given to over-sampling when the thr.rel parameter is NA. When O increases the number of examples to include during the over-sampling step also increases. Default to 0.5.

U — A number expressing the importance given to under-sampling when the thr.rel parameter is NA. When U increases, the number of examples selected during the under-sampling step also increases. Defaults to 0.5.

Value

The function returns a data frame with the new data set resulting from the application of the importance sampling strategy.

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

See Also

RandUnderRegress, RandOverRegress

Examples

```
library(DMwR)
data(algae)
clean.algae <- algae[complete.cases(algae), ]
# defining a threshold on the relevance
IS.ext <-ImpSampRegress(a7~., clean.algae, rel = "auto",
                        thr.rel = 0.7, C.perc = "extreme")
IS.bal <-ImpSampRegress(a7~., clean.algae, rel = "auto", thr.rel = 0.7,
                        C.perc = "balance")
myIS <-ImpSampRegress(a7~., clean.algae, rel = "auto", thr.rel = 0.7,
                      C.perc = list(0.2, 6))
# neither threshold nor C.perc defined
IS.auto <- ImpSampRegress(a7~., clean.algae, rel = "auto")
```

NCLClassif — *Neighborhood Cleaning Rule (NCL) algorithm for multiclass imbalanced problems*

Description

This function handles imbalanced classification problems using the Neighborhood Cleaning Rule (NCL) method.

Usage

```
NCLClassif(form, dat, k = 3, dist = "Euclidean", p = 2, Cl = "smaller")
```

Arguments

form	A formula describing the prediction problem.
dat	A data frame containing the original imbalanced data set.
k	A number indicating the number of nearest neighbors to use.
dist	A character string indicating which distance metric to use when determining the k nearest neighbors. See the details. Defaults to "Euclidean".
p	A number indicating the value of p if the "p-norm" distance is chosen. Only necessary to define if a "p-norm" is chosen in the dist argument. See details.
Cl	A character vector indicating which classes should be under-sampled. Defaults to "smaller" meaning that all "smaller"" classes are the most important and therefore only examples from the remaining classes should be removed. The user may define a subset of the existing classes in which this technique will be applied.

Details

dist **parameter:** The parameter dist allows the user to define the distance metric to be used in the neighbors computation. Although the default is the Euclidean distance, other metrics are available. This allows the computation of distances in data sets with, for instance, both nominal and numeric features. The options available for the distance functions are as follows:

- for data with only numeric features: "Manhattan", "Euclidean", "Canberra", "Chebyshev", "p-norm";

- for data with only nominal features: "Overlap";

- for dealing with both nominal and numeric features: "HEOM", "HVDM".

When the "p-norm" is selected for the dist parameter, it is also necessary to define the value of parameter p. The value of parameter p sets which "p-norm" will be used. For instance, if p is set to 1, the "1-norm" (or Manhattan distance) is used, and if p is set to 2, the "2-norm" (or Euclidean distance) is applied. For more details regarding the distance functions implemented in UBL package please see the package vignettes.

NCL algorithm: The NCL algorithm includes two phases. In the first phase the ENN algorithm is used to under-sample the examples whose class label is not in Cl. Then, a second step is performed which aims at further clean the neighborhood of the examples in Cl. To achieve this, the k nearest neighbors of examples in Cl are scanned. An example is removed if all the previous neighbors have a class label which is not in Cl, and if the example belongs to a class which is larger than half of the smaller class in Cl. In either steps the examples with class labels in Cl are always maintained.

Value

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

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

References

J. Laurikkala. (2001). *Improving identification of difficult small classes by balancing class distribution*. Artificial Intelligence in Medicine, pages 63-66.

See Also

ENNClassif

Examples

```
# generate a small imbalanced data set
ir <- iris[-c(90:135), ]
# apply NCL method with different metrics, number of neighbors and classes
ir.M1 <- NCLClassif(Species~., ir, k = 3, dist = "Manhattan", Cl = "smaller")
ir.Def <- NCLClassif(Species~., ir)
ir.Ch <- NCLClassif(Species~., ir, k = 7, dist = "Chebyshev", Cl = "virginica")
ir.Eu <- NCLClassif(Species~., ir, k = 5, Cl = c("versicolor", "virginica"))
# check the results
summary(ir$Species)
summary(ir.M1$Species)
summary(ir.Def$Species)
summary(ir.Ch$Species)
summary(ir.Eu$Species)
```

---

neighbours *Computation of nearest neighbours using a selected distance function.*

Description

This function allows to obtain the nearest neighbours of each example in a data set using a distance function selected by the user.

Usage

```
neighbours(tgt, dat, dist, p=2, k)
```

Arguments

tgt	The column of the problem target variable.
dat	A data frame containing the problem data.
dist	A character string specifying the distance function to use in the nearest neighbours evaluation.
p	An optional parameter that is only required if the distance function selected in parameter dist is "p-norm".
k	The number of nearest neighbours to return for each example.

Details

Several distance function are implemented in UBL package. The goal of having such a diversity of distance functions is to provide the users more flexibility regarding the distance used and also to provide distance fucntions that are able to deal with nominal and numeric features. The options available for the distance functions are as follows:

data with only numeric features: "Manhattan", "Euclidean", "Canberra", "Chebyshev", "p-norm";

data with only nominal features: "Overlap";

data with both nominal and numeric features: "HEOM", "HVDM".

When the "p-norm" is selected for the `dist` parameter, it is also necessary to define the value of parameter `p`. The value of parameter `p` sets which "p-norm" will be used. For instance, if `p` is set to 1, the "1-norm" (or Manhattan distance) is used, and if `p` is set to 2, the "2-norm" (or Euclidean distance) is applied. For more details regarding the distance functions implemented in UBL package please see the package vignettes.

Value

The function returns a matrix with the indexes of the k nearest neighbours for each example in the data set.

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Wilson, D.R. and Martinez, T.R. (1997). *Improved heterogeneous distance functions.* Journal of artificial intelligence research, pp.1-34.

See Also

distances

Examples

```
## Not run:
data(ImbC)
# determine the 2 nearest neighbours of each example in ImbC data set
# using the "HVDM" distance function.
neig1 <- neighbours(3, ImbC, "HVDM", k=2)

# now using the "HEOM" distance function
neig2 <- neighbours(3, ImbC, "HEOM", k=2)

# check the differences
head(neig1)
head(neig2)

## End(Not run)
```

`OSSClassif` *One-sided selection strategy for handling multiclass imbalanced problems.*

Description

This function performs an adapted one-sided selection strategy for multiclass imbalanced problems.

Usage

```
OSSClassif(form, dat, dist = "Euclidean", p = 2, Cl = "smaller", start = "CNN")
```

Arguments

`form`	A formula describing the prediction problem.
`dat`	A data frame containing the original imbalanced data set.
`dist`	A character string indicating which distance metric to use when determining the k nearest neighbors. See the details. Defaults to "Euclidean".
`p`	A number indicating the value of p if the "p-norm" distance is chosen. Only necessary to define if a "p-norm" is chosen in the `dist` argument. See details.
`Cl`	A character vector indicating which are the most important classes. Defaults to "smaller" which means that the smaller classes are automatically determined. In this case, all the smaller classes are those with a frequency below (nr.examples)/(nr.classes). With the selection of option "smaller" those classes are the ones considered important for the user.
`start`	A string which determines which strategy (CNN or Tomek links) should be performed first. The existing options are "CNN" and "Tomek". The first one, "CNN", which is the default, means that CNN strategy will be performed first and Tomek links are applied after. On the other hand, if `start` is set to "Tomek" then the reverse order is applied (first Tomek links and after CNN strategy).

Details

`dist` **parameter:** The parameter `dist` allows the user to define the distance metric to be used in the neighbors computation. Although the default is the Euclidean distance, other metrics are available. This allows the computation of distances in data sets with, for instance, both nominal and numeric features. The options available for the distance functions are as follows:

- for data with only numeric features: "Manhattan", "Euclidean", "Canberra", "Chebyshev", "p-norm";

- for data with only nominal features: "Overlap";

- for dealing with both nominal and numeric features: "HEOM", "HVDM".

When the "p-norm" is selected for the `dist` parameter, it is also necessary to define the value of parameter `p`. The value of parameter `p` sets which "p-norm" will be used. For instance, if `p` is set to 1, the "1-norm" (or Manhattan distance) is used, and if `p` is set to 2, the "2-norm" (or Euclidean distance) is applied. For more details regarding the distance functions implemented in UBL package please see the package vignettes.

Value

The function returns a data frame with the new data set resulting from the application of the selected OSS strategy.

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Kubat, M. & Matwin, S. (1997). *Addressing the Curse of Imbalanced Training Sets: One-Sided Selection* Proc. of the 14th Int. Conf. on Machine Learning, Morgan Kaufmann, 179-186.

Batista, G. E.; Prati, R. C. & Monard, M. C. (2004). *A study of the behavior of several methods for balancing machine learning training data* ACM SIGKDD Explorations Newsletter, ACM, 6, 20-29

See Also

TomekClassif, CNNClassif

Examples

```
## Not run:
library(DMwR)
data(algae)
clean.algae <- algae[complete.cases(algae), ]
alg1 <- OSSClassif(season~., clean.algae, dist = "HVDM",
                   Cl = c("spring", "summer"))
alg2 <- OSSClassif(season~., clean.algae, dist = "HEOM",
                   Cl = c("spring", "summer"), start = "Tomek")
alg3 <- OSSClassif(season~., clean.algae, dist = "HVDM", start = "CNN")
alg4 <- OSSClassif(season~., clean.algae, dist = "HVDM", start = "Tomek")
alg5 <- OSSClassif(season~., clean.algae, dist = "HEOM", Cl = "winter")
summary(alg1$season)
summary(alg2$season)
summary(alg3$season)
summary(alg4$season)
summary(alg5$season)

## End(Not run)
```

---

phi *Relevance function.*

Description

This function allows to obtain the relevance function values on a set of target variable values given the interpolating points.

Usage

```
phi(y, control.parms)
```

Arguments

y	The target variable values of the problem.
control.parms	A named list supplied by the phi.control function with the parameters needed for obtaining the relevance values.

Details

The phi function specifies the regions of interest in the target variable. It does so by performing a Monotone Cubic Spline Interpolation over a set of maximum and minimum relevance points. The notion of relevance can be associated with rarity. Nonetheless, this notion may depend on the domain experts knowledge.

Value

The function returns the relevance values.

Author(s)

Rita Ribeiro <rpribeiro@dcc.fc.up.pt>, Paula Branco <paobranco@gmail.com>, and Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Ribeiro, R., 2011. Utility-based regression (Doctoral dissertation, PhD thesis, Dep. Computer Science, Faculty of Sciences - University of Porto).

Fritsch, F.N. and Carlson, R.E., 1980. Monotone piecewise cubic interpolation. SIAM Journal on Numerical Analysis, 17(2), pp.238-246.

See Also

phi.control

Examples

```
# example of a relevance function where the extremes are the important values.
data(morley)
# the target variable
y <- morley$Speed

phiF.args <- phi.control(y,method="extremes",extr.type="both")
y.phi <- phi(y, control.parms=phiF.args)
plot(y, y.phi)
```

`phi.control` *Estimation of parameters used for obtaining the relevance function.*

Description

This function allows to obtain the parameters of the relevance function (phi). The parameters can be obtained using one of the following methods: "extremes" or "range". If the selected method is "extremes", the distribution of the target variable values is used to assign more importance to the most extreme values according to the boxplot. If the selected method is "range", a matrix should be provided defining the important and unimportant values (see section details).

Usage

```
phi.control(y, method="extremes", extr.type="both", coef=1.5, control.pts=NULL)
```

Arguments

`y`	The target variable values.
`method`	"extremes" (default) or "range".
`extr.type`	parameter needed for method "extremes" to specify which type of extremes are to be considered relevant: "high", "low" or "both"(default).
`coef`	parameter needed for method "extremes" to specify how far the wiskers extend to the most extreme data point in the boxplot. The default is 1.5.
`control.pts`	parameter needed for method "range" to specify the interpolating points to the relevance function (phi). It should be a matrix with three columns. The first column represents the y value, the second column represents the corresponding relevance value (phi(y)) in [0,1], and the third optional column represents the corresponding relevance value derivative (phi'(y)).

Details

The method "extremes" uses the target variable distribution to automatically determine the most relevant values. This method uses the boxplot to automatically derive a matrix with the interpolating points for the relevance function (phi). According to the `extr.type` parameter it assigns maximum relevance to: only the "high" extremes, only the "low" extremes or "both". In the latter case, it assigns maximum relevance to "both" types of extremes if they exist. If "both" is selected and only one type of extremes is present, then only the existing extremes are considered.

The method "range" uses the `control.pts` matrix provided by the user to define the interpolating points for the relevance function (phi). The values supplied in the third column (phi derivative) of the matrix are only indicative, meaning that they will be adjusted afterwards by the relevance function (phi) to create a smooth continuous function.

Value

The function returns a list with the parameters needed for obtaining and evaluating the relevance function (phi).

Author(s)

Rita Ribeiro <rpribeiro@dcc.fc.up.pt>, Paula Branco <paobranco@gmail.com> and Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Ribeiro, R., 2011. Utility-based regression (Doctoral dissertation, PhD thesis, Dep. Computer Science, Faculty of Sciences - University of Porto).

See Also

phi

Examples

```
data(morley)
# the target variable
y <- morley$Speed

# the target variable has "low" and "high"extremes
boxplot(y)

## using method "extremes" considering that
## "both" extremes are important
phiF.argsB <- phi.control(y,method="extremes",extr.type="both")
y.phiB <- phi(y, control.parms=phiF.argsB)
plot(y, y.phiB)

## using method "extremes" considering that only the
## "high" extremes are relevant
phiF.argsH <- phi.control(y,method="extremes",extr.type="high")
y.phiH <- phi(y, control.parms=phiF.argsH)
plot(y, y.phiH)


## using method "range" to choose the important values:
rel <- matrix(0,ncol=3,nrow=0)
rel <- rbind(rel,c(700,0,0))
rel <- rbind(rel,c(800,1,0))
rel <- rbind(rel,c(900,0,0))
rel <- rbind(rel,c(1000,1,0))
rel
phiF.argsR <- phi.control(y,method="range",control.pts=rel)
y.phiR <- phi(y, control.parms=phiF.argsR)

plot(y, y.phiR)
```

`RandOverClassif` *Random over-sampling for imbalanced classification problems*

Description

This function performs a random over-sampling strategy for imbalanced multiclass problems. Essentially, a percentage of cases of the class(es) selected by the user are randomly over-sampled by the introduction of replicas of examples. Alternatively, the strategy can be applied to either balance all the existing classes or to "smoothly invert" the frequency of the examples in each class.

Usage

```
RandOverClassif(form, dat, C.perc = "balance", repl = TRUE)
```

Arguments

`form`	A formula describing the prediction problem.
`dat`	A data frame containing the original imbalanced data set.
`C.perc`	A named list containing each class name and the corresponding over-sampling percentage, greater than or equal to 1, where 1 means that no over-sampling is to be applied in the corresponding class. The user may indicate only the classes where he wants to apply random over-sampling. For instance, a percenatge of 2 means that, in the changed data set, the number of examples of that class are doubled. Alternatively, this parameter can be set to "balance" (the default) or "extreme", cases where the over-sampling percentages are automatically estimated. The "balance" option tries to balance all the existing classes while the "extreme" option inverts the classes original frequencies.
`repl`	A boolean value controlling the possibility of having repetition of examples when choosing the examples to repeat in the over-sampled data set. Defaults to TRUE because this is a necessary condition if the selected percentage is greater than 2. This parameter is only important when the over-sampling percentage is between 1 and 2. In this case, it controls if all the new examples selected from a given class can be repeated or not.

Details

This function performs a random over-sampling strategy for dealing with imbalanced multiclass problems. The new examples included in the new data set are replicas of the examples already present in the original data set.

Value

The function returns a data frame with the new data set resulting from the application of the random over-sampling strategy.

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

See Also

RandUnderClassif

Examples

```
library(DMwR)
data(algae)
# classes spring and winter remain unchanged
C.perc = list(autumn = 2, summer = 1.5, spring = 1)
myover.algae <- RandOverClassif(season~., algae, C.perc)
oveBalan.algae <- RandOverClassif(season~., algae, "balance")
oveInvert.algae <- RandOverClassif(season~., algae, "extreme")

library(MASS)
data(cats)
myover.cats <- RandOverClassif(Sex~., cats, list(M = 1.5))
oveBalan.cats <- RandOverClassif(Sex~., cats, "balance")
oveInvert.cats <- RandOverClassif(Sex~., cats, "extreme")

# learn a model and check results with original and over-sampled data
library(rpart)
idx <- sample(1:nrow(cats), as.integer(0.7 * nrow(cats)))
tr <- cats[idx, ]
ts <- cats[-idx, ]

ct0 <- rpart(Sex ~ ., tr)
preds0 <- predict(ct0, ts, type = "class")
new.cats <- RandOverClassif(Sex~., tr, "balance")
ct1 <- rpart(Sex ~ ., new.cats)
preds1 <- predict(ct1, ts, type = "class")
table(preds0, ts$Sex)
table(preds1, ts$Sex)
```

RandOverRegress *Random over-sampling for imbalanced regression problems*

Description

This function performs a random over-sampling strategy for imbalanced regression problems. Basically a percentage of cases of the "class(es)" (bumps above a relevance threshold defined) selected by the user are randomly over-sampled. Alternatively, it can either balance all the existing "classes" (the default) or it can "smoothly invert" the frequency of the examples in each class.

Usage

```
RandOverRegress(form, dat, rel = "auto", thr.rel = 0.5,
                C.perc = "balance", repl = TRUE)
```

Arguments

form	A formula describing the prediction problem.
dat	A data frame containing the original imbalanced data set.
rel	The relevance function which can be automatically ("auto") determined (the default) or may be provided by the user through a matrix with the interpolating points.
thr.rel	A number indicating the relevance threshold above which a case is considered as belonging to the rare "class".
C.perc	A list containing the over-sampling percentage/s to apply to all/each "class" (bump) obtained with the relevance threshold. Replicas of the examples are are randomly added in each "class". If only one percentage is provided this value is reused in all the "classes" that have values above the relevance threshold. A different percentage can be provided to each "class". In this case, the percentages should be provided in ascending order of target variable value. The over-sampling percentage(s), should be numbers above 1, meaning that the important cases (cases above the threshold) are over-sampled by the corresponding percentage. If the number 1 is provided then those examples are not changed. Alternatively, C.perc parameter may be set to "balance" or "extreme", cases where the over-sampling percentages are automatically estimated to either balance or invert the frequencies of the examples in the "classes" (bumps).
repl	A boolean value controlling the possibility of having repetition of examples when choosing the examples to repeat in the over-sampled data set. Defaults to TRUE because this is a necessary condition if the selected percentage is greater than 2. This parameter is only important when the over-sampling percentage is between 1 and 2. In this case, it controls if all the new examples selected from a given "class" can be repeated or not.

Details

This function performs a random over-sampling strategy for dealing with imbalanced regression problems. The new examples included in the new data set are randomly selected replicas of the examples already present in the original data set.

Value

The function returns a data frame with the new data set resulting from the application of the random over-sampling strategy.

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

See Also

RandUnderRegress

Examples

```
data(morley)

C.perc = list(2, 4)
myover <- RandOverRegress(Speed~., morley, C.perc=C.perc)
Bal <- RandOverRegress(Speed~., morley, C.perc= "balance")
Ext <- RandOverRegress(Speed~., morley, C.perc= "extreme")

library(DMwR)
data(algae)
clean.algae <- algae[complete.cases(algae), ]
# all automatic
ROB <-RandOverRegress(a7~., clean.algae)
# user defined percentage for the only existing extreme (high)
myRO <-RandOverRegress(a7~., clean.algae, rel = "auto", thr.rel = 0.7,
                       C.perc = list(5))

# check the results
plot(clean.algae[,c(1,ncol(clean.algae))])
plot(ROB[,c(1,ncol(clean.algae))])
plot(myRO[,c(1,ncol(clean.algae))])
```

RandUnderClassif *Random under-sampling for imbalanced classification problems*

Description

This function performs a random under-sampling strategy for imbalanced multiclass problems. Essentially, a percentage of cases of the class(es) selected by the user are randomly removed. Alternatively, the strategy can be applied to either balance all the existing classes or to "smoothly invert" the frequency of the examples in each class.

Usage

```
RandUnderClassif(form, dat, C.perc = "balance", repl = FALSE)
```

Arguments

form	A formula describing the prediction problem.
dat	A data frame containing the original imbalanced data set.

C.perc	A named list containing each class name and the corresponding under-sampling percentage, between 0 and 1, where 1 means that no under-sampling is to be applied in the corresponding class. The user may indicate only the classes where he wants to apply random under-sampling. For instance, a percentage of 0.2 means that, in the changed data set, the class is reduced to 20% of its original size. Alternatively, this parameter can be set to "balance" (the defualt) or "extreme", cases where the under-sampling percentages are automatically estimated. The "balance" option tries to balance all the existing classes while the "extreme" option inverts the classes original frequencies.
repl	A boolean value controlling the possibility of having repetition of examples in the under-sampled data set. Defaults to FALSE.

Details

This function performs a random under-sampling strategy for dealing with imbalanced multiclass problems. The examples removed are randomly selected among the examples belonging to each class containing the normal cases. The user can chose one or more classes to be under-sampled.

Value

The function returns a data frame with the new data set resulting from the application of the random under-sampling strategy.

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

See Also

RandOverClassif

Examples

```
library(DMwR)
data(algae)
C.perc = list(autumn = 1, summer = 0.9, winter = 0.4)
# classes autumn and spring remain unchanged

myunder.algae <- RandUnderClassif(season~., algae, C.perc)
undBalan.algae <- RandUnderClassif(season~., algae, "balance")
undInvert.algae <- RandUnderClassif(season~., algae, "extreme")

library(MASS)
data(cats)
myunder.cats <- RandUnderClassif(Sex~., cats, list(M = 0.8))
undBalan.cats <- RandUnderClassif(Sex~., cats, "balance")
undInvert.cats <- RandUnderClassif(Sex~., cats, "extreme")


# learn a model and check results with original and under-sampled data
```

```
library(rpart)
idx <- sample(1:nrow(cats), as.integer(0.7*nrow(cats)))
tr <- cats[idx, ]
ts <- cats[-idx, ]

idx <- sample(1:nrow(cats), as.integer(0.7*nrow(cats)))
tr <- cats[idx, ]
ts <- cats[-idx, ]
ct0 <- rpart(Sex ~ ., tr)
preds0 <- predict(ct0, ts, type = "class")
new.cats <- RandUnderClassif(Sex~., tr, "balance")
ct1 <- rpart(Sex ~ ., new.cats)
preds1 <- predict(ct1, ts, type = "class")

table(preds0, ts$Sex)
#  preds0  F  M
#      F  9  3
#      M  7 25


table(preds1, ts$Sex)
# preds1  F  M
#      F 13  4
#      M  3 24
```

RandUnderRegress *Random under-sampling for imbalanced regression problems*

Description

This function performs a random under-sampling strategy for imbalanced regression problems. Essentially, a percentage of cases of the "class(es)" (bumps below a relevance threshold defined) selected by the user are randomly removed. Alternatively, the strategy can be applied to either balance all the existing "classes"" or to "smoothly invert" the frequency of the examples in each "class".

Usage

```
RandUnderRegress(form, dat,  rel = "auto", thr.rel = 0.5,
                 C.perc = "balance", repl = FALSE)
```

Arguments

form	A formula describing the prediction problem.
dat	A data frame containing the original imbalanced data set.
rel	The relevance function which can be automatically ("auto") determined (the default) or may be provided by the user through a matrix with interpolating points.
thr.rel	A number indicating the relevance threshold below which a case is considered as belonging to the normal "class".

`C.perc` A list containing the under-sampling percentage/s to apply to all/each "class" (bump) obtained with the relevance threshold. Examples are randomly removed from the "class(es)". If only one percentage is provided this value is reused in all the "classes" that have values below the relevance threshold. A different percentage can be provided to each "class". In this case, the percentages should be provided in ascending order of target variable value. The under-sampling percentage(s), should be a number below 1, meaning that the normal cases (cases below the threshold) are under-sampled by the corresponding percentage. If the number 1 is provided then those examples are not changed. Alternatively, `C.perc` parameter may be set to "balance" or "extreme", cases where the under-sampling percentages are automatically estimated to either balance or invert the frequencies of the examples in the "classes" (bumps).

`repl` A boolean value controlling the possibility of having repetition of examples in the under-sampled data set. Defaults to FALSE.

Details

This function performs a random under-sampling strategy for dealing with imbalanced regression problems. The examples removed are randomly selected among the examples belonging to the normal "class(es)" (bump of relevance below the threshold defined). The user can chose one or more bumps to be under-sampled.

Value

The function returns a data frame with the new data set resulting from the application of the random under-sampling strategy.

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

See Also

RandOverRegress

Examples

```
data(morley)

C.perc = list(0.5)
myUnd <- RandUnderRegress(Speed~., morley, C.perc=C.perc)
Bal <- RandUnderRegress(Speed~., morley, C.perc= "balance")
Ext <- RandUnderRegress(Speed~., morley, C.perc= "extreme")
```

SmoteClassif *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

```
SmoteClassif(form, dat, C.perc = "balance", k = 5, repl = FALSE,
             dist = "Euclidean", p = 2)
```

Arguments

form	A formula describing the prediction problem
dat	A data frame containing the original (unbalanced) data set
C.perc	A named list containing the percentage(s) of under- or/and over-sampling to apply to each class. The over-sampling percentage is a number above 1 while the under-sampling percentage should be a number below 1. If the number 1 is provided for a given class then that class remains unchanged. Alternatively it may be "balance" (the default) or "extreme", cases where the sampling percentages are automatically estimated either to balance the examples between the minority and majority classes or to invert the distribution of examples across the existing classes transforming the majority classes into minority and vice-versa.
k	A number indicating the number of nearest neighbors that are used to generate the new examples of the minority class(es).
repl	A boolean value controlling the possibility of having repetition of examples when performing under-sampling by selecting among the majority class(es) examples.
dist	A character string indicating which distance metric to use when determining the k nearest neighbors. See the details. Defaults to "Euclidean".
p	A number indicating the value of p if the "p-norm" distance is chosen. Only necessary to define if a "p-norm" is chosen in the dist argument. See details.

Details

dist **parameter:** The parameter dist allows the user to define the distance metric to be used in the neighbors computation. Although the default is the Euclidean distance, other metrics are available. This allows the computation of distances in data sets with, for instance, both nominal and numeric features. The options available for the distance functions are as follows:

- for data with only numeric features: "Manhattan", "Euclidean", "Canberra", "Chebyshev", "p-norm";

- for data with only nominal features: "Overlap";

- for dealing with both nominal and numeric features: "HEOM", "HVDM".

When the "p-norm" is selected for the `dist` parameter, it is also necessary to define the value of parameter `p`. The value of parameter `p` sets which "p-norm" will be used. For instance, if `p` is set to 1, the "1-norm" (or Manhattan distance) is used, and if `p` is set to 2, the "2-norm" (or Euclidean distance) is applied. For more details regarding the distance functions implemented in UBL package please see the package vignettes.

Smote algorithm: 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 parameter `C.perc` controls the amount of over-sampling and under-sampling applied and can be automatically estimated either to balance or invert the distribution of examples across the different classes. The parameter `k` controls the number of neighbors used to generate new synthetic examples.

Value

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

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and 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.

See Also

`RandUnderClassif`, `RandOverClassif`

Examples

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

## now using SMOTE to create a more "balanced problem"
newData <- SmoteClassif(Species ~ ., dat, C.perc = list(common = 1,rare = 6))
table(newData$Species)
```

```
## Checking visually the created data
par(mfrow = c(1, 2))
plot(dat[, 1], dat[, 2], pch = 19 + as.integer(dat[, 3]),
     main = "Original Data")
plot(newData[, 1], newData[, 2], pch = 19 + as.integer(newData[, 3]),
     main = "SMOTE'd Data")


# automatically balancing the data maintaining the total number of examples
datBal <- SmoteClassif(Species ~ ., dat, C.perc = "balance")
table(datBal$Species)

# automatically inverting the original distribution of examples
datExt <- SmoteClassif(Species ~ ., dat, C.perc = "extreme")
table(datExt$Species)


 library(DMwR)
 data(algae)
 clean.algae <- algae[complete.cases(algae),]
 C.perc = list(autumn = 2, summer = 1.5, winter = 0.9)
 # class spring remains unchanged
 # In this case it is necessary to define a distance function that
 # is able to deal with both nominal and numeric features
 mysmote.algae <- SmoteClassif(season~., clean.algae, C.perc, dist = "HEOM")
 # the distance function may be HVDM
 smoteBalan.algae <- SmoteClassif(season~., clean.algae, "balance",
                                  dist = "HVDM")
 smoteExtre.algae <- SmoteClassif(season~., clean.algae, "extreme",
                                  dist = "HVDM")

  library(MASS)
  data(cats)
  mysmote.cats <- SmoteClassif(Sex~., cats, list(M = 0.8, F = 1.8))
  smoteBalan.cats <- SmoteClassif(Sex~., cats, "balance")
  smoteExtre.cats <- SmoteClassif(Sex~., cats, "extreme")
```

SmoteRegress *SMOTE algorithm for imbalanced regression problems*

Description

This function handles imbalanced regression problems using the SMOTE method. Namely, it can generate a new "SMOTEd" data set that addresses the problem of imbalanced domains.

Usage

```
SmoteRegress(form, dat, rel = "auto", thr.rel = 0.5, C.perc = "balance",
                        k = 5, repl = FALSE, dist = "Euclidean", p = 2)
```

Arguments

form	A formula describing the prediction problem
dat	A data frame containing the original (unbalanced) data set
rel	The relevance function which can be automatically ("auto") determined (the default) or may be provided by the user through a matrix.
thr.rel	A number indicating the relevance threshold above which a case is considered as belonging to the rare "class".
C.perc	A list containing the percentage(s) of under- or/and over-sampling to apply to each "class" (bump) obtained with the threshold. The percentages should be provided in ascending order of target variable value. The percentages are applied in this order to the "classes" (bumps) obtained through the threshold. The over-sampling percentage, a number above 1, means that the examples in that bump are increased by this percentage. The under-sampling percentage, a number below 1, means that the cases in the corresponding bump are under-sampled by this percentage. If the number 1 is provided then those examples are not changed. Alternatively it may be "balance" (the default) or "extreme", cases where the sampling percentages are automatically estimated.
k	A number indicating the number of nearest neighbors to consider as the pool from where the new generated examples are generated.
repl	A boolean value controlling the possibility of having repetition of examples when performing under-sampling by selecting among the "normal" examples.
dist	A character string indicating which distance metric to use when determining the k nearest neighbors. See the details. Defaults to "Euclidean".
p	A number indicating the value of p if the "p-norm" distance is chosen. Only necessary to define if a "p-norm" is chosen in the dist argument. see details.

Details

dist **parameter:** The parameter dist allows the user to define the distance metric to be used in the neighbors computation. Although the default is the Euclidean distance, other metrics are available. This allows the computation of distances in data sets with, for instance, both nominal and numeric features. The options available for the distance functions are as follows:

- for data with only numeric features: "Manhattan", "Euclidean", "Canberra", "Chebyshev", "p-norm";

- for data with only nominal features: "Overlap";

- for dealing with both nominal and numeric features: "HEOM".

When the "p-norm" is selected for the dist parameter, it is also necessary to define the value of parameter p. The value of parameter p sets which "p-norm" will be used. For instance, if p is set to 1, the "1-norm" (or Manhattan distance) is used, and if p is set to 2, the "2-norm" (or Euclidean distance) is applied. For more details regarding the distance functions implemented in UBL package please see the package vignettes.

SmoteR algorithm: Imbalanced domains cause problems to many learning algorithms. These problems are characterized by the uneven proportion of cases that are available for certain ranges of the target variable which are the most important to the user.

SMOTE (Chawla et. al. 2002) is a well-known algorithm for classification tasks 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 data set. SmoteR is a variant of SMOTE algorithm proposed by Torgo et al. (2013) to address the problem of imbalanced domains in regression tasks. This function uses the parameters `rel` and `thr.rel`, a relevance function and a relevance threshold for distinguishing between the normal and rare cases.

The parameter `C.perc` controls the amount of over-sampling and under-sampling applied and can be automatically estimated either to balance or invert the distribution of examples across the different bumps. The parameter `k` controls the number of neighbors used to generate new synthetic examples.

Value

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

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and 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, Luis and Ribeiro, Rita P and Pfahringer, Bernhard and Branco, Paula (2013). *SMOTE for Regression*. Progress in Artificial Intelligence, Springer,378-389.

See Also

`RandUnderRegress`, `RandOverRegress`

Examples

```
ir <- iris[-c(95:130), ]
mysmote1.iris <- SmoteRegress(Sepal.Width~., ir, dist = "HEOM",
                              C.perc=list(0.5,2.5))
mysmote2.iris <- SmoteRegress(Sepal.Width~., ir, dist = "HEOM",
                              C.perc = list(0.2, 4), thr.rel = 0.8)
smoteBalan.iris <- SmoteRegress(Sepal.Width~., ir, dist = "HEOM",
                              C.perc = "balance")
smoteExtre.iris <- SmoteRegress(Sepal.Width~., ir, dist = "HEOM",
                              C.perc = "extreme")

# checking visually the results
plot(sort(ir$Sepal.Width))
plot(sort(smoteExtre.iris$Sepal.Width))
```

```
# using a relevance function provided by the user
rel <- matrix(0, ncol = 3, nrow = 0)
rel <- rbind(rel, c(2, 1, 0))
rel <- rbind(rel, c(3, 0, 0))
rel <- rbind(rel, c(4, 1, 0))

sP.ir <- SmoteRegress(Sepal.Width~., ir, rel = rel, dist = "HEOM",
                      C.perc = list(4, 0.5, 4))
```

`TomekClassif` *Tomek links for imbalanced classification problems*

Description

This function uses Tomek links to perform under-sampling for handling imbalanced multiclass problems. Tomek links are broken by removing one or both examples forming the link.

Usage

```
TomekClassif(form, dat, dist = "Euclidean", p = 2, Cl = "all", rem = "both")
```

Arguments

`form`	A formula describing the prediction problem.
`dat`	A data frame containing the original imbalanced data set.
`dist`	A character string indicating which distance metric to use when determining the k nearest neighbors. See the details. Defaults to "Euclidean".
`p`	A number indicating the value of p if the "p-norm" distance is chosen. Only necessary to define if a "p-norm" is chosen in the `dist` argument. See details.
`Cl`	A character vector indicating which classes should be under-sampled. Defaults to "all" meaning that examples from all existing classes can be removed. The user may also specify a subset of classes for which tomek links should be removed.
`rem`	A character string indicating if both examples forming the Tomek link are to be removed, or if only the example from the larger class should be discarded. In the first case this parameter should be set to "both" and in the second case should be set to "maj".

Details

`dist` **parameter:** The parameter `dist` allows the user to define the distance metric to be used in the neighbors computation. Although the default is the Euclidean distance, other metrics are available. This allows the computation of distances in data sets with, for instance, both nominal and numeric features. The options available for the distance functions are as follows:

- for data with only numeric features: "Manhattan", "Euclidean", "Canberra", "Chebyshev", "p-norm";

- for data with only nominal features: "Overlap";

- for dealing with both nominal and numeric features: "HEOM", "HVDM".

When the "p-norm" is selected for the `dist` parameter, it is also necessary to define the value of parameter `p`. The value of parameter `p` sets which "p-norm" will be used. For instance, if `p` is set to 1, the "1-norm" (or Manhattan distance) is used, and if `p` is set to 2, the "2-norm" (or Euclidean distance) is applied. For more details regarding the distance functions implemented in UBL package please see the package vignettes.

Tomek method: This function performs an under-sampling strategy based on the notion of Tomek links for imbalanced multiclass problems. Two examples form a Tomek link if they are each other closest neighbors and they have different class labels.

The under-sampling procedure can be performed in two different ways. When detected the Tomek links, the examples of both classes can be removed, or the Tomek link can be broken by removing only one of the examples (traditionally the one belonging to the majority class). This function also includes these two procedures. Moreover, it allows for the user to identify in which classes under-sampling should be applied. These two aspects are controlled by the `Cl` and `rem` parameters. The `Cl` parameter is used to express the classes that can be under-sampled and its default is "all" (all existing classes are candidates for having examples removed). The parameter `rem` indicates if the Tomek link is broken by removing both examples ("both") or by removing only the example belonging to the more populated class between the two existing in the Tomek link.

Note that the options for `Cl` and `rem` may "disagree". In those cases, the preference is given to the `Cl` options once the user choose that specific set of classes to under-sample and not the other ones (even if the defined classes are not the larger ones). This means that, when making a decision on how many and which examples will be removed the first criteria used will be the `Cl` definition .

For a better clarification of the impact of the options selected for Cl and rem parameters we now provide some possible scenarios and the expected behavior:

1) `Cl` is set to one class which is neither the more nor the less frequent, and `rem` is set to "maj". The expected behavior is the following: - if a Tomek link exists connecting the largest class and another class(not included in `Cl`): no example is removed; - if a Tomek link exists connecting the larger class and the class defined in `Cl`: the example from the `Cl` class is removed (because the user expressly indicates that only examples from class `Cl` should be removed);

2) `Cl` includes two classes and `rem` is set to "both". This function will do the following: - if a Tomek link exists between an example with class in `Cl` and another example with class not in `Cl`, then, only the example with class in `Cl` is removed; - if the Tomek link exists between two examples with classes in `Cl`, then, both are removed.

3) `Cl` includes two classes and `rem` is set to "maj". The behavior of this function is the following: -if a Tomek link exists connecting two classes included in `Cl`, then only the example belonging to the more populated class is removed; -if a Tomek link exists connecting an example from a class included in `Cl` and another example whose class is not in `Cl` and is the largest class, then, no example is removed.

Value

The function returns a list containing a data frame with the new data set resulting from the application of the Tomek link strategy defined, and the indexes of the examples removed.

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Tomek, I. (1976). *Two modifications of CNN* IEEE Trans. Syst. Man Cybern., 769-772

See Also

OSSClassif, CNNClassif

Examples

```
library(DMwR)
data(algae)
clean.algae <- algae[complete.cases(algae), ]
alg.HVDM1 <- TomekClassif(season~., clean.algae, dist = "HVDM",
                          Cl = c("winter", "spring"), rem = "both")
alg.HVDM2 <- TomekClassif(season~., clean.algae, dist = "HVDM", rem = "maj")

# removes only examples from class summer which are the
# majority class in the link
alg.EuM <- TomekClassif(season~., clean.algae, dist = "HEOM",
                        Cl = "summer", rem = "maj")

# removes only examples from class summer in every link they appear
alg.EuB <- TomekClassif(season~., clean.algae, dist = "HEOM",
                        Cl = "summer", rem = "both")

summary(clean.algae$season)
summary(alg.HVDM1[[1]]$season)
summary(alg.HVDM2[[1]]$season)
summary(alg.EuM[[1]]$season)
summary(alg.EuB[[1]]$season)

# check which were the indexes of the examples removed in alg.EuM
alg.EuM[[2]]
```

UtilInterpol *Utility surface obtained through methods for spatial interpolation of points.*

Description

This function uses spatial interpolation methods for obtaining the utility surface. It depends on a set of points provided by the user and on a method selected for interpolation. The available interpolation methods are: bilinear, splines, idw and krige. Check the details section for more on these methods.

Usage

```
UtilInterpol(trues, preds, type = c("utility", "cost", "benefit"), control.parms,
          minds, maxds, m.pts, method = c("bilinear", "splines", "idw", "krige"),
            visual = FALSE, eps = 0.1, full.output = FALSE)
```

Arguments

trues	A vector of true target variable values. Can be NULL. See details section.
preds	A vector with corresponding predicted values for the trues provided. Can be NULL. See details section.
type	A character specifying the type of surface that is being interpolated. It can be set to either "utility", "cost" or "benefit". When set to "cost" we assume that the diagonal of the surface (where y=y.pred) is zero. Therefore, in this case, the user doesn't need to set the control.parms parameter.
control.parms	These parameters are necessary for utility and benefit surfaces. control.parms can be obtained with a call to function phi.control. This provides a list with the parameters used for defining the relevance function phi. The points provided through these parameters are used for interpolating the utility surface because the relevance function matches the diagonal of the utility, i.e., the relevance function phi corresponds to the utility of accurate predictions (y = y.pred). Alternatively, the user may build the control.parms list. When the user selects a cost surface, control.parms can simply be NULL. In this case, we assume that the surface diagonal is zero. If control.parms are not NULL, then specified points are used. See examples section.
minds	The lower bound of the target variable considered for interpolation. A new minds value may be necessary when trues and/or preds provided have lower values than minds. This is handled by extrapolation and a warning is issued (see details).
maxds	The upper bound of the target variable considered for interpolation. A new maxds value may be necessary when trues and/or preds provided have values higher than maxds. This is handled by extrapolation and a warning is issued (see details).
m.pts	A 3-column matrix with interpolating points for the off-diagonal cases (i.e., y != y.pred), provided by the user. The first column has the y value, the second column the y.pred value and the third column has the corresponding utility value. At least, the off diagonal domain boundary points (i.e., points (minds, maxds, util) and (maxds, minds, util)) must be provided in this matrix. Moreover, the points provided through this parameter must be in [minds, maxds] range.
method	A character indicating which interpolation method should be used. Can be one of: "bilinear", "splines", "idw" or "krige". See details section for a description of the available methods.
visual	Logical. If TRUE a plot of the utility surface isometrics obtained and the points provided is displayed. If FALSE (the default) no image is plotted.
eps	Numeric value for the precision considered during the interpolation. Defaults to 0.1. Only relevant if a plot is displayed, or when full.output is set to TRUE. See details section.

`full.output` Logical. If FALSE (the default) only the results from points provided through parameters trues and preds are returned. If TRUE a matrix with the utility of all points in domain (considering the eps provided) are returned. See details section.

Details

`method` **parameter:** The parameter `method` allows the user to select from a set of interpolation methods. The available methods are as follows:

- `bilinear`: local fitting of a polynomial surface of degree 1 obtained through loess function of stats package.

- `splines`: multilevel B-splines interpolation method obtained through MBA R package.

- `idw`: inverse distance weighted interpolation obtained through R package gstat.

- `krige`: automatic kriging obtained using automap R package.

extrapolation: when trues or preds provided are outside the range [minds, maxds] the function performs an extrapolation of the domain. To achieve this, four new points are added that extend the initial target variable domain ([minds, maxds]). This extrapolation is performed as follows:

- first: determine inc.fac, the distance necessary to increase (the largest value needed increase the axes to include all trues and preds provided);

- second: define the new target variable domain ([minds - inc.fac, maxds + inc.fac]);

- third: add two new diagonal points evaluating the relevance function on these new points (i.e. add (minds-inc.fac, minds-nc.fac, phi(minds-inc.fac, minds.inc.fac)) and (maxds+inc.fac, maxds+inc.fac, phi(maxds+inc.fac, maxds+inc.fac)));

- fourth: add two new off-diagonal points using the new min and max values of the domain and the utility provided by the user for the two mandatory points (minds, maxds) and (maxds, minds).

In order to avoid this extrapolation, the user must ensure that the values provided in trues and preds vectors are inside the [minds, maxds] range provided.

`full.output` **parameter:** This parameter is used to select which utility values are returned. There are two options for this parameter:

- FALSE: This means that the user is only interested in obtaining the utility surface values of some points (y, y.pred). In this case, the y and y.pred should be provided through parameters trues and preds and the function returns a vector with the utility for the corresponding points.

- TRUE: The user is interested in obtaining the utility surface values on a grid of equally spaced values of the target variable domain. In this case, there is no need for specifying parameters trues and preds, because the goal is not to observe the utility of these points. Parameters trues and preds can be set to NULL in this case. The function returns a lXl matrix with the utility of all points in a grid defined as follows. The l equally spaced points are a sequence that starts at minds-0.01, ends at maxds+0.01 and are incremented by eps value.

Value

The function returns a vector with utility of the points provided through the vectors trues and preds.

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

See Also

phi.control, UtilOptimRegress

Examples

```
## Not run:
# examples with a utility surface
data(Boston, package = "MASS")

tgt <- which(colnames(Boston) == "medv")
sp <- sample(1:nrow(Boston), as.integer(0.7*nrow(Boston)))
train <- Boston[sp,]
test <- Boston[-sp,]

control.parms <- phi.control(Boston[,tgt], method="extremes", extr.type="both")
# the boundaries of the domain considered
minds <- min(Boston[,tgt])-5
maxds <- max(Boston[,tgt])+5

# build m.pts to include at least the utility of the
# points (minds, maxds) and (maxds, minds)
m.pts <- matrix(c(minds, maxds, -1, maxds, minds, 0),
                byrow=TRUE, ncol=3)

trues <- test[,tgt]
library(randomForest)
model <- randomForest(medv~., train)
preds <- predict(model, test)

resLIN <- UtilInterpol(trues, preds, type="util", control.parms, minds, maxds, m.pts,
                         method = "bilinear", visual=TRUE)
resIDW <- UtilInterpol(trues, preds, type="util", control.parms, minds, maxds, m.pts,
                        method = "idw", visual=TRUE)
resSPL <- UtilInterpol(trues, preds, type="util", control.parms, minds, maxds, m.pts,
                        method = "spl", visual=TRUE)
resKRIGE <- UtilInterpol(trues, preds, type="util", control.parms, minds, maxds, m.pts,
                          method = "krige", visual=TRUE)

# examples with a cost surface
data(Boston, package = "MASS")

tgt <- which(colnames(Boston) == "medv")
sp <- sample(1:nrow(Boston), as.integer(0.7*nrow(Boston)))
train <- Boston[sp,]
test <- Boston[-sp,]

# the boundaries of the domain considered
```

```
minds <- min(Boston[,tgt])-5
maxds <- max(Boston[,tgt])+5

# build m.pts to include at least the utility of the
# points (minds, maxds) and (maxds, minds)
m.pts <- matrix(c(minds, maxds, 5, maxds, minds, 20),
                byrow=TRUE, ncol=3)

trues <- test[,tgt]

# train a model and predict on test set
library(randomForest)
model <- randomForest(medv~., train)
preds <- predict(model, test)

costLIN <- UtilInterpol(trues, preds, type="cost", control.parms=NULL, minds, maxds, m.pts,
                         method = "bilinear", visual=TRUE )

costSPL <- UtilInterpol(trues, preds, type="cost", control.parms=NULL, minds, maxds, m.pts,
                        method = "spl", visual=TRUE)

costKRIGE <- UtilInterpol(trues, preds, type="cost", control.parms=NULL, minds, maxds, m.pts,
                          method = "krige", visual=TRUE)

costIDW <- UtilInterpol(trues, preds, type="cost", control.parms=NULL, minds, maxds, m.pts,
                        method = "idw", visual=TRUE)


# if the user has a cost matrix and wants to specify the control.parms:
my.pts <- matrix(c(0, 0, 0, 10, 0, 0, 20, 0, 0, 45, 0, 0), byrow=TRUE, ncol=3)
control.parms <- phi.control(trues, method="range", control.pts = my.pts)

costLIN <- UtilInterpol(trues, preds, type="cost", control.parms=control.parms,
                        minds, maxds, m.pts, method = "bilinear", visual=TRUE )


# first trues and preds
trues[1:5]
preds[1:5]
trues[1:5]-preds[1:5]

# first cost results on these predictions for cost surface costIDW
costIDW[1:5]
# a summary of these prediction costs:
summary(costIDW)

#example with a benefit surface

# define control.parms either by defining a list with 3 named elements
# or by calling phi.control function with method range and passing
# the selected control.pts
control.parms <- list(method="range", npts=5,
                      control.pts=c(0,1,0,10,5,0.5,20,10,0.5,30,30,0,50,30,0))
```

```
m.pts <- matrix(c(minds, maxds, 0, maxds, minds, 0),
                byrow=TRUE, ncol=3)

benLIN <- UtilInterpol(trues, preds, type="ben", control.parms, minds, maxds, m.pts,
                       method = "bilinear", visual=TRUE)
benIDW <- UtilInterpol(trues, preds, type="ben", control.parms, minds, maxds, m.pts,
                       method = "idw", visual=TRUE)
benSPL <- UtilInterpol(trues, preds, type="ben", control.parms, minds, maxds, m.pts,
                       method = "spl", visual=TRUE)
benKRIGE <- UtilInterpol(trues, preds, type="ben", control.parms, minds, maxds, m.pts,
                         method = "krige", visual=TRUE)

## End(Not run)
```

`UtilOptimClassif` *Optimization of predictions utility, cost or benefit for classification problems.*

Description

This function determines the optimal predictions given a utility, cost or benefit matrix for the selected learning algorithm. The learning algorithm must provide probabilities for the problem classes. If the matrix provided is of type utility or benefit a maximization process is carried out. If the user provides a cost matrix, then a minimization process is applied.

Usage

```
UtilOptimClassif(form, train, test, mtr, type = "util",
                 learner = NULL, learner.pars=NULL, predictor="predict",
                 predictor.pars=NULL)
```

Arguments

`form`	A formula describing the prediction problem.
`train`	A data.frame with the training data.
`test`	A data.frame with the test data.
`mtr`	A matrix, specifying the utility, cost, or benefit values associated to accurate predictions and misclassification errors. It can be either a cost matrix, a benefit matrix or a utility matrix. The corresponding type should be set in parameter type. The matrix must be always provided with the true class in the rows and the predicted class in the columns.
`type`	The type of mtr provided. Can be set to: "utility"(default), "cost" or "benefit".
`learner`	Character specifying the learning algorithm to use. It is required for the selected learner to provide class probabilities.
`learner.pars`	A named list containing the parameters of the learning algorithm.
`predictor`	Character specifying the predictor selected (defaults to "predict").
`predictor.pars`	A named list with the predictor selected parameters.

Value

The function returns a vector with the predictions for the test data optimized using the matrix provided.

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Elkan, C., 2001, August. The foundations of cost-sensitive learning. In International joint conference on artificial intelligence (Vol. 17, No. 1, pp. 973-978). LAWRENCE ERLBAUM ASSOCIATES LTD.

See Also

UtilOptimRegress, EvalClassifMetrics

Examples

```
# the synthetic data set provided with UBL package for classification
data(ImbC)
sp <- sample(1:nrow(ImbC), round(0.7*nrow(ImbC)))
train <- ImbC[sp, ]
test <- ImbC[-sp,]

# example with a utility matrix
# define a utility matrix (true class in rows and pred class in columns)
matU <- matrix(c(0.2, -0.5, -0.3, -1, 1, -0.9, -0.9, -0.8, 0.9), byrow=TRUE, ncol=3)

library(e1071) # for the naiveBayes classifier

resUtil <- UtilOptimClassif(Class~., train, test, mtr = matU, type="util",
                      learner = "naiveBayes",
                      predictor.pars = list(type="raw", threshold = 0.01))

# learning a standard model without maximizing utility
model <- naiveBayes(Class~., train)
resNormal <- predict(model, test, type="class", threshold = 0.01)
# Check the difference in the total utility of the results
EvalClassifMetrics(test$Class, resNormal, mtr=matU, type= "util")
EvalClassifMetrics(test$Class, resUtil, mtr=matU, type= "util")


#example with a cost matrix
# define a cost matrix (true class in rows and pred class in columns)
matC <- matrix(c(0, 0.5, 0.3, 1, 0, 0.9, 0.9, 0.8, 0), byrow=TRUE, ncol=3)
resUtil <- UtilOptimClassif(Class~., train, test, mtr = matC, type="cost",
                          learner = "naiveBayes",
                          predictor.pars = list(type="raw", threshold = 0.01))
```

```
# learning a standard model without minimizing the costs
model <- naiveBayes(Class~., train)
resNormal <- predict(model, test, type="class")
# Check the difference in the total utility of the results
EvalClassifMetrics(test$Class, resNormal, mtr=matC, type= "cost")
EvalClassifMetrics(test$Class, resUtil, mtr=matC, type= "cost")


#example with a benefit matrix
# define a benefit matrix (true class in rows and pred class in columns)
matB <- matrix(c(0.2, 0, 0, 0, 1, 0, 0, 0, 0.9), byrow=TRUE, ncol=3)

resUtil <- UtilOptimClassif(Class~., train, test, mtr = matB, type="ben",
                            learner = "naiveBayes",
                            predictor.pars = list(type="raw", threshold = 0.01))

# learning a standard model without maximizing benefits
model <- naiveBayes(Class~., train)
resNormal <- predict(model, test, type="class", threshold = 0.01)
# Check the difference in the total utility of the results
EvalClassifMetrics(test$Class, resNormal, mtr=matB, type= "ben")
EvalClassifMetrics(test$Class, resUtil, mtr=matB, type= "ben")

table(test$Class,resNormal)
table(test$Class,resUtil)
```

`UtilOptimRegress` *Optimization of predictions utility, cost or benefit for regression problems.*

Description

This function determines the optimal predictions given a utility, cost or benefit surface. This surface is obtained through a specified strategy with some parameters. For determining the optimal predictions an estimation of the conditional probability density function is performed for each test case. If the surface provided is of type utility or benefit a maximization process is carried out. If the user provides a cost surface, then a minimization is performed.

Usage

```
UtilOptimRegress(form, train, test, type = "util", strat = "interpol",
                 strat.parms = list(method = "bilinear"), control.parms, m.pts,
                 minds, maxds, eps = 0.1)
```

Arguments

form	A formula describing the prediction problem.
train	A data.frame with the training data.
test	A data.frame with the test data.
type	A character specifying the type of surface provided. Can be one of: "utility", "cost" or "benefit". Defaults to "utility".
strat	A character determining the strategy for obtaining the surface of the problem. For now, only the interpolation strategy is available (the default).
strat.parms	A named list containing the parameters necessary for the strategy previously specified. For the interpolation strategy (the default and only strategy available for now), it is required that the user specifies wich method sould be used for interpolating the points.
control.parms	A named list with the control.parms defined through the function phi.control. These parameters stablish the diagonal of the surface provided. If the type of surface defined is "cost" this parameter can be set to NULL, because in this case we assume that the accurate prediction, i.e., points in the diagonal of the surface have zero cost. See examples.
m.pts	A matrix with 3-columns, with interpolation points specifying the utility, cost or benefit of the surface. The points sould be in the off-diagonal of the surface, i.e., the user should provide points where y != y.pred. The first column must have the true value (y), the second column the corresponding prediction (y.pred) and the third column sets the utility cost or benefit of that point (y, y.pred). The user should define as many points as possible. The minimum number of required points are two. More specifically, the user must always set the surface values of at least the points (minds, maxds) and (maxds, minds). See minds and maxds description.
maxds	The numeric upper bound of the target variable considered.
minds	The numeric lower bound of the target variable considered.
eps	Numeric value for the precision considered during the interpolation. Defaults to 0.1.

Details

The optimization process carried out by this function uses a method for conditional density estimation proposed by Rau M.M et al.(2015). Code for conditional density estimation (available on github https://github.com/MarkusMichaelRau/OrdinalClassification) kindly contributed by M. M. Rau with changes made by P.Branco. The optimization is achieved generalizing the method proposed by Elkan (2001) for classification tasks. In regression, this process involves determining, for each test case, the maximum integral (for utility or benefit surfaces, or the minimum if we have a cost surface) of the product of the conditional density function estimated and either the utility, the benefit or the cost surface. The optimal prediction for a case q is given by: $y^*(q) = argmax[z] \int pdf(y|q).U(y,z)dy$, where pdf(y|q) is the conditional densitiy estimation for case q, and U(y,z) is the utility, benefit or cost surface evaluated on the true value y and predictied value z.

Value

The function returns a vector with the predictions for the test data optimized using the surface provided.

Author(s)

Paula Branco <paobranco@gmail.com>, Rita Ribeiro <rpribeiro@dcc.fc.up.pt> and Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Rau, M.M., Seitz, S., Brimioulle, F., Frank, E., Friedrich, O., Gruen, D. and Hoyle, B., 2015. Accurate photometric redshift probability density estimation-method comparison and application. Monthly Notices of the Royal Astronomical Society, 452(4), pp.3710-3725.

Elkan, C., 2001, August. The foundations of cost-sensitive learning. In International joint conference on artificial intelligence (Vol. 17, No. 1, pp. 973-978). LAWRENCE ERLBAUM ASSOCIATES LTD.

See Also

phi.control, UtilOptimClassif, UtilInterpol

Examples

```
## Not run:
#Example using a utility surface:
data(Boston, package = "MASS")

tgt <- which(colnames(Boston) == "medv")
sp <- sample(1:nrow(Boston), as.integer(0.7*nrow(Boston)))
train <- Boston[sp,]
test <- Boston[-sp,]

control.parms <- phi.control(Boston[,tgt], method="extremes", extr.type="both")
# the boundaries of the domain considered
minds <- min(train[,tgt])
maxds <- max(train[,tgt])

# build m.pts to include at least (minds, maxds) and (maxds, minds) points
# m.pts must only contain points in [minds, maxds] range.
m.pts <- matrix(c(minds, maxds, -1, maxds, minds, -1),
                byrow=TRUE, ncol=3)

pred.res <- UtilOptimRegress(medv~., train, test, type = "util", strat = "interpol",
                             strat.parms=list(method = "bilinear"),
                             control.parms = control.parms,
                             m.pts = m.pts, minds = minds, maxds = maxds)

eval.util <- EvalRegressMetrics(test$medv, pred.res$optim, pred.res$utilRes,
                                thr=0.8, control.parms = control.parms)
```

```
# train a normal model
model <- randomForest(medv~.,train)
normal.preds <- predict(model, test)

#obtain the utility of the new points (trues, preds)
NormalUtil <- UtilInterpol(test$medv, normal.preds, type="util",
                           control.parms = control.parms,
                           minds, maxds, m.pts, method = "bilinear")
#check the performance
eval.normal <- EvalRegressMetrics(test$medv, normal.preds, NormalUtil,
                                  thr=0.8, control.parms = control.parms)

#check both results
eval.util
eval.normal


#check visually both predictions and the surface used
UtilInterpol(test$medv, normal.preds, type = "util", control.parms = control.parms,
                           minds, maxds, m.pts, method = "bilinear", visual=TRUE)

points(test$medv, normal.preds, col="green")
points(test$medv, pred.res$optim, col="blue")

# another example now using points interpolation with splines
data(algae,package="DMwR")
ds <- algae[complete.cases(algae[,1:12]),1:12]
tgt <- which(colnames(ds) == "a1")
sp <- sample(1:nrow(ds), as.integer(0.7*nrow(ds)))
train <- ds[sp,]
test <- ds[-sp,]

control.parms <- phi.control(ds[,tgt], method="extremes", extr.type="both")

# the boundaries of the domain considered
minds <- min(train[,tgt])
maxds <- max(train[,tgt])

# build m.pts to include at least (minds, maxds) and (maxds, minds) points
m.pts <- matrix(c(minds, maxds, -1, maxds, minds, -1),
                byrow=TRUE, ncol=3)

pred.res <- UtilOptimRegress(a1~., train, test, type = "util", strat = "interpol",
                             strat.parms=list(method = "splines"),
                             control.parms = control.parms,
                             m.pts = m.pts, minds = minds, maxds = maxds)

# check the predictions
plot(test$a1, pred.res$optim)
# assess the performance
eval.util <- EvalRegressMetrics(test$a1, pred.res$optim, pred.res$utilRes,
                                thr=0.8, control.parms = control.parms)
#
```

```
# train a normal model
model <- randomForest(a1~.,train)
normal.preds <- predict(model, test)

#obtain the utility of the new points (trues, preds)
NormalUtil <- UtilInterpol(test$medv, normal.preds, type = "util",
                           control.parms = control.parms,
                           minds, maxds, m.pts, method="splines")
#check the performance
eval.normal <- EvalRegressMetrics(test$medv, normal.preds, NormalUtil,
                                  thr=0.8, control.parms = control.parms)

eval.util
eval.normal

# observe the utility surface with the normal preds
UtilInterpol(test$a1, normal.preds, type="util", control.parms = control.parms,
             minds, maxds, m.pts, method="splines", visual=TRUE)
# add the optim preds
points(test$a1, pred.res$optim, col="green")

# Example using a cost surface:
data(Boston, package = "MASS")

tgt <- which(colnames(Boston) == "medv")
sp <- sample(1:nrow(Boston), as.integer(0.7*nrow(Boston)))
train <- Boston[sp,]
test <- Boston[-sp,]

# if using interpolation methods for COST surface, the control.parms can be set to NULL
# the boundaries of the domain considered
minds <- min(train[,tgt])
maxds <- max(train[,tgt])

# build m.pts to include at least (minds, maxds) and (maxds, minds) points
m.pts <- matrix(c(minds, maxds, 5, maxds, minds, 20),
                byrow=TRUE, ncol=3)

pred.res <- UtilOptimRegress(medv~., train, test, type = "cost", strat = "interpol",
                             strat.parms = list(method = "bilinear"),
                             control.parms = NULL,
                             m.pts = m.pts, minds = minds, maxds = maxds)

# check the predictions
plot(test$medv, pred.res$optim)

# assess the performance
eval.util <- EvalRegressMetrics(test$medv, pred.res$optim, pred.res$utilRes,
                                type="cost", maxC = 20)
#
# train a normal model
model <- randomForest(medv~.,train)
normal.preds <- predict(model, test)
```

```
#obtain the utility of the new points (trues, preds)
NormalUtil <- UtilInterpol(test$medv, normal.preds, type="cost", control.parms = NULL,
                           minds, maxds, m.pts, method="bilinear")
#check the performance
eval.normal <- EvalRegressMetrics(test$medv, normal.preds, NormalUtil,
                                  type="cost", maxC = 20)

eval.normal
eval.util

# check visually the surface and the predictions
UtilInterpol(test$medv, normal.preds, type="cost", control.parms = NULL,
                           minds, maxds, m.pts, method="bilinear",
                           visual=TRUE)

points(test$medv, pred.res$optim, col="blue")


## End(Not run)
```

Index