Package 'mlquantify'

September 23, 2020

Type Package

Title Algorithms for Class Distribution Estimation

Version 0.1.4

Maintainer Andre Maletzke <andregustavom@gmail.com>

Description Quantification is a prominent machine learning task that has received an increasing amount of attention in the last years. The objective is to predict the class distribution of a data sample. This package is a collection of machine learning algorithms for class distribution estimation. This package include algorithms from different paradigms of quantification. These methods are described in the paper: A. Maletzke, W. Hassan, D. dos Reis, and G. Batista. The importance of the test set size in quantification assessment. In Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI20, pages 2640–2646, 2020. <doi:10.24963/ijcai.2020/366>.

License GPL (>= 2.0)

Encoding UTF-8

LazyData true

NeedsCompilation no

Author Andre Maletzke [aut, cre],
Everton Cherman [ctb],
Denis dos Reis [ctb],
Gustavo Batista [ths]

RoxygenNote 7.1.1

BugReports https://github.com/andregustavom/mlquantify/issues

URL https://github.com/andregustavom/mlquantify

Imports caret, randomForest, stats

Suggests CORElearn

Depends R (>= 3.5.0)

Repository CRAN

Date/Publication 2020-09-23 09:20:08 UTC

R topics documented:

`ACC` *Adjusted Classify and Count*

Description

It quantifies events based on testing scores using the Adjusted Classify and Count (ACC) method. ACC is an extension of CC, applying a correction rate based on the true and false positive rates (`tpr` and `fpr`).

Usage

```
ACC(test, TprFpr, thr=0.5)
```

Arguments

`test`	a numeric `vector` containing the score estimated for the positive class from each test set instance.
`TprFpr`	a `data.frame` of true positive (`tpr`) and false positive (`fpr`) rates estimated on training set, using the function `getTPRandFPRbyThreshold()`.
`thr`	threshold value according to the `tpr` and `fpr` were learned. Default is `0.5`.

Value

the class distribution in the test set.

References

Forman, G. (2006, August). Quantifying trends accurately despite classifier error and class imbalance. In ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 157-166).<doi.org/10.1145/1150402.1150423>.

Examples

```
library(randomForest)
library(caret)
cv <- createFolds(aeAegypti$class, 3)
tr <- aeAegypti[cv$Fold1,]
validation <- aeAegypti[cv$Fold2,]
ts <- aeAegypti[cv$Fold3,]

# -- Getting a sample from ts with 80 positive and 20 negative instances --
ts_sample <- rbind(ts[sample(which(ts$class==1),80),],
                   ts[sample(which(ts$class==2),20),])
scorer <- randomForest(class~., data=tr, ntree=500)
scores <- cbind(predict(scorer, validation, type = c("prob")), validation$class)
TprFpr <- getTPRandFPRbyThreshold(scores)
test.scores <- predict(scorer, ts_sample, type = c("prob"))
ACC(test = test.scores[,1], TprFpr = TprFpr)
```

---

aeAegypti *Males and Females Aedes Aegypti data from Maletzke (2019)*

Description

Contains events generated by a laser sensor to capture the flight dynamism of insects. It is a binary dataset compose by events from *Aedes Aegypti* Female and Male.

Usage

```
data(aeAegypti)
```

Format

The data set aeAegypti is a data frame of 1800 observations of 9 variables. Each event is described by the wing beat frequency (wbf), and the frequencies of the first six harmonics obtained when either female or male *Aedes Aegypti* mosquito cross an optical sensor's line-of-sigh. Both male (class = 2) and female (class = 1) of class factor.

Details

The aeAegypti dataset is a subset of widely data collection effort involving more than one million instances for 20 different insect species including mosquitoes, house flies, bees, and wasps. The whole dataset was collected varying the temperature from 22ºC to 34ºC (increments of 2ºC) and the humidity from 20

Author(s)

Andre Maletzke <andregustavom@gmail.com>

References

Maletzke, A. G. (2019). Binary quantification in non-stationary scenarios. Doctoral Thesis, Instituto de Ciências Matemáticas e de Computação, University of São Paulo, São Carlos. Retrieved 2020-07-21, from www.teses.usp.br. <doi.org/10.11606/T.55.2020.tde-19032020-091709>

Moreira dos Reis, D., Maletzke, A., Silva, D. F., & Batista, G. E. (2018). Classifying and counting with recurrent contexts. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 1983-1992). <doi.org/10.1145/3219819.3220059>

CC *Classify and Count*

Description

It quantifies events based on testing scores, applying the Classify and Count (CC). CC is the simplest quantification method that derives from classification (Forman, 2005).

Usage

```
CC(test, thr=0.5)
```

Arguments

test	a numeric vector containing the score estimated for the positive class from each test set instance.
thr	a numeric value indicating the decision threshold. A value between 0 and 1 (default = 0.5)

Value

the class distribution in the test set.

References

Forman, G. (2005). Counting positives accurately despite inaccurate classification. In European Conference on Machine Learning. Springer, Berlin, Heidelberg.<doi.org/10.1007/11564096_55>.

Examples

```
library(randomForest)
library(caret)
cv <- createFolds(aeAegypti$class, 2)
tr <- aeAegypti[cv$Fold1,]
ts <- aeAegypti[cv$Fold2,]

# -- Getting a sample from ts with 80 positive and 20 negative instances --
ts_sample <- rbind(ts[sample(which(ts$class==1),80),],
                   ts[sample(which(ts$class==2),20),])
scorer <- randomForest(class~., data=tr, ntree=500)
test.scores <- predict(scorer, ts_sample, type = c("prob"))
CC(test = test.scores[,1])
```

DyS *DyS Framework*

Description

DyS is a framework for quantification data based on mixture models method. It quantifies events based on testing scores, applying the DyS framework proposed by Maletzke et al. (2019). It also works with several similarity functions.

Usage

```
DyS(p.score, n.score, test, measure="topsoe", bins=seq(2,20,2), err=1e-5)
```

Arguments

`p.score`	a numeric `vector` of positive scores estimated either from a validation set or from a cross-validation method.
`n.score`	a numeric `vector` of negative scores estimated either from a validation set or from a cross-validation method.
`test`	a numeric `vector` containing the score estimated for the positive class from each test set instance.
`measure`	measure used to compare the mixture histogram against the histogram obtained from the test set. Several functions can be used (Default: `"topsoe"`, `"euclidean"`, `"jensen_difference"`, `"prob_symm"`, `"taneja"`, `"ord"`).
`bins`	a numeric `vector` of number of bins used to construct the histogram for representing the score distribution. (default: `seq(2,20,2)`).
`err`	a numeric value defining the accepted error for the ternary search (default: `1e5`).

Value

the class distribution in the test.

References

Maletzke, A., Reis, D., Cherman, E., & Batista, G. (2019). DyS: a Framework for Mixture Models in Quantification. in Proceedings of the The Thirty-Third AAAI Conference on Artificial Intelligence, ser. AAAI'19, 2019. <doi.org/10.1609/aaai.v33i01.33014552>.

Examples

```
library(randomForest)
library(caret)
cv <- createFolds(aeAegypti$class, 3)
tr <- aeAegypti[cv$Fold1,]
validation <- aeAegypti[cv$Fold2,]
ts <- aeAegypti[cv$Fold3,]

# -- Getting a sample from ts with 80 positive and 20 negative instances --
ts_sample <- rbind(ts[sample(which(ts$class==1),80),],
                   ts[sample(which(ts$class==2),20),])
scorer <- randomForest(class~., data=tr, ntree=500)
scores <- cbind(predict(scorer, validation, type = c("prob")), validation$class)
test.scores <- predict(scorer, ts_sample, type = c("prob"))
DyS(p.score = scores[scores[,3]==1,1], n.score = scores[scores[,3]==2,1],
test = test.scores[,1])
```

EMQ *Expectation-Maximization Quantification*

Description

This method is an instance of the well-known algorithm for finding maximum-likelihood estimates of the model's parameters. It quantifies events based on testing scores, applying the Expectation Maximization for Quantification (EMQ) method proposed by Saerens et al. (2002).

Usage

```
EMQ(train, test, it=5, e=NULL)
```

Arguments

`train`	a `data.frame` of the labeled set.
`test`	a numeric `matrix` of scores predicted from each test set instance. First column must be the positive score.
`it`	maximum number of iteration steps (default `5`).
`e`	a numeric value for the stop threshold (default `NULL`). If the difference between two consecutive steps is lower or equal than `e`, the iterative process will be stopped. If `e` is null then the iteration phase is defined by the `it` parameter.

Value

the class distribution in the test set.

References

Saerens, M., Latinne, P., & Decaestecker, C. (2002). Adjusting the outputs of a classifier to new a priori probabilities: a simple procedure. Neural computation.<doi.org/10.1162/089976602753284446>.

Examples

```
library(randomForest)
library(caret)
cv <- createFolds(aeAegypti$class, 3)
tr <- aeAegypti[cv$Fold1,]
validation <- aeAegypti[cv$Fold2,]
ts <- aeAegypti[cv$Fold3,]

# -- Getting a sample from ts with 80 positive and 20 negative instances --
ts_sample <- rbind(ts[sample(which(ts$class==1),80),],
                   ts[sample(which(ts$class==2),20),])
scorer <- randomForest(class~., data=tr, ntree=500)
scores <- cbind(predict(scorer, validation, type = c("prob")), validation$class)
test.scores <- predict(scorer, ts_sample, type = c("prob"))
EMQ(train=tr, test=test.scores)
```

`getHist` *Get a normalized histogram from score values.*

Description

Get a normalized histogram from score values.

Usage

```
getHist(scores, nbins)
```

Arguments

`scores`	a numeric vector containing the scores.
`nbins`	number of histogram bins.

Value

normalized histogram.

Author(s)

Andre Maletzke <andregustavom@gmail.com>

Examples

```
positive_scores <- runif(200)^0.3
negative_scores <- 1 - runif(200)^0.3
getHist(positive_scores, 20)
getHist(negative_scores, 20)
```

`getTPRandFPRbyThreshold` *Estimates true and false positive rates*

Description

This function provides the true and false positive rates (`tpr` and `fpr`) for a range of thresholds.

Usage

```
getTPRandFPRbyThreshold(validation_scores, label_pos = 1, thr_range = seq(0.01,0.99,0.01))
```

Arguments

`validation_scores`	`data.frame` scores estimated from the training set. It should be comprised of three columns (1. positive scores; 2. negative scores; 3.class).
`label_pos`	numeric value or factor indicating the positive label.
`thr_range`	a numerical `vector` of thresholds, ranged between 0 and 1. Default: `seq(0.01,0.99,0.01)`.

Value

`data.frame` where each row has both (`tpr` and `fpr`) rates for each threshold value. This function varies the threshold from 0.01 to 0.99 with increments 0.01.

Author(s)

Everton Cherman <evertoncherman@gmail.com>

Andre Maletzke <andregustavom@gmail.com>

Examples

```
library(randomForest)
library(caret)
cv <- createFolds(aeAegypti$class, 2)
tr <- aeAegypti[cv$Fold1,]
validation <- aeAegypti[cv$Fold2,]
scorer <- randomForest(class~., data=tr, ntree=500)
scores <- cbind(predict(scorer, validation, type = c("prob")), validation$class)
TprFpr <- getTPRandFPRbyThreshold(scores)
```

HDy_LP *HDy with Laplace smoothing*

Description

It computes the class distribution using the HDy algorithm proposed by González-Castro et al. (2013) with Laplace smoothing (Maletzke et al. (2019)).

Usage

```
HDy_LP(p.score, n.score, test)
```

Arguments

p.score	a numeric vector of positive scores estimated either from a validation set or from a cross-validation method.
n.score	a numeric vector of negative scores estimated either from a validation set or from a cross-validation method.
test	a numeric vector containing the score estimated for the positive class from each test set instance.

Value

the class distribution in the test set.

Author(s)

Andre Maletzke <andregustavom@gmail.com>

References

González-Castro, V., Alaíz-Rodriguez, R., & Alegre, E. (2013). Class distribution estimation based on the Hellinger distance. Information Sciences.<doi.org/10.1016/j.ins.2012.05.028>

Maletzke, A., Reis, D., Cherman, E., & Batista, G. (2019). DyS: a Framework for Mixture Models in Quantification. in Proceedings of the The Thirty-Third AAAI Conference on Artificial Intelligence, ser. AAAI'19, 2019. <doi.org/10.1609/aaai.v33i01.33014552>.

Examples

```
library(randomForest)
library(caret)
cv <- createFolds(aeAegypti$class, 3)
tr <- aeAegypti[cv$Fold1,]
validation <- aeAegypti[cv$Fold2,]
ts <- aeAegypti[cv$Fold3,]

# -- Getting a sample from ts with 80 positive and 20 negative instances --
ts_sample <- rbind(ts[sample(which(ts$class==1),80),],
```

```
                  ts[sample(which(ts$class==2),20),])
scorer <- randomForest(class~., data=tr, ntree=500)
scores <- cbind(predict(scorer, validation, type = c("prob")), validation$class)
test.scores <- predict(scorer, ts_sample, type = c("prob"))
HDy_LP(p.score = scores[scores[,3]==1,1], n.score=scores[scores[,3]==2,1],
test=test.scores[,1])
```

KUIPER *Quantification method based on Kuiper's test*

Description

It quantifies events based on testing scores, applying an adaptation of the Kuiper's test for quantification problems.

Usage

```
KUIPER(p.score, n.score, test)
```

Arguments

p.score	a numeric vector of positive scores estimated either from a validation set or from a cross-validation method.
n.score	a numeric vector of negative scores estimated either from a validation set or from a cross-validation method.
test	a numeric vector containing the score estimated for the positive class from each test set instance.

Value

the class distribution in the test set.

Author(s)

Denis dos Reis <denismr@gmail.com>

Examples

```
library(randomForest)
library(caret)
cv <- createFolds(aeAegypti$class, 3)
tr <- aeAegypti[cv$Fold1,]
validation <- aeAegypti[cv$Fold2,]
ts <- aeAegypti[cv$Fold3,]

# -- Getting a sample from ts with 80 positive and 20 negative instances --
ts_sample <- rbind(ts[sample(which(ts$class==1),80),],
                  ts[sample(which(ts$class==2),20),])
```

```
scorer <- randomForest(class~., data=tr, ntree=500)
scores <- cbind(predict(scorer, validation, type = c("prob")), validation$class)
test.scores <- predict(scorer, ts_sample, type = c("prob"))
KUIPER(p.score = scores[scores[,3]==1,1], n.score = scores[scores[,3]==2,1],
test = test.scores[,1])
```

MAX *Threshold selection method*

Description

It quantifies events based on testing scores, applying MAX method, according to Forman (2006). Same as T50, but it sets the threshold where `tpr−fpr` is maximized.

Usage

```
MAX(test, TprFpr)
```

Arguments

`test`	a numeric `vector` containing the score estimated for the positive class from each test set instance.
`TprFpr`	a `data.frame` of true positive (`tpr`) and false positive (`fpr`) rates estimated on training set, using the function `getTPRandFPRbyThreshold()`.

Value

the class distribution in the test set.

References

Forman, G. (2006, August). Quantifying trends accurately despite classifier error and class imbalance. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 157-166).<doi.org/10.1145/1150402.1150423>.

Examples

```
library(randomForest)
library(caret)
cv <- createFolds(aeAegypti$class, 3)
tr <- aeAegypti[cv$Fold1,]
validation <- aeAegypti[cv$Fold2,]
ts <- aeAegypti[cv$Fold3,]

# -- Getting a sample from ts with 80 positive and 20 negative instances --
ts_sample <- rbind(ts[sample(which(ts$class==1),80),],
                   ts[sample(which(ts$class==2),20),])
scorer <- randomForest(class~., data=tr, ntree=500)
scores <- cbind(predict(scorer, validation, type = c("prob")), validation$class)
```

```
TprFpr <- getTPRandFPRbyThreshold(scores)
test.scores <- predict(scorer, ts_sample, type = c("prob"))
MAX(test=test.scores[,1], TprFpr=TprFpr)
```

MKS *Mixable Kolmogorov Smirnov*

Description

It quantifies events based on testing scores, applying the Mixable Kolmogorov Smirnov (MKS) method proposed by Maletzke et al. (2019).

Usage

```
MKS(p.score, n.score, test)
```

Arguments

p.score	a numeric vector of positive scores estimated either from a validation set or from a cross-validation method.
n.score	a numeric vector of negative scores estimated either from a validation set or from a cross-validation method.
test	a numeric vector containing the score estimated for the positive class from each test set instance.

Value

the class distribution in the test set.

References

Maletzke, A., Reis, D., Cherman, E., & Batista, G. (2019). DyS: a Framework for Mixture Models in Quantification. in Proceedings of the The Thirty-Third AAAI Conference on Artificial Intelligence, ser. AAAI'19, 2019.<doi.org/10.1609/aaai.v33i01.33014552>

Examples

```
library(randomForest)
library(caret)
cv <- createFolds(aeAegypti$class, 3)
tr <- aeAegypti[cv$Fold1,]
validation <- aeAegypti[cv$Fold2,]
ts <- aeAegypti[cv$Fold3,]

# -- Getting a sample from ts with 80 positive and 20 negative instances --
ts_sample <- rbind(ts[sample(which(ts$class==1),80),],
                   ts[sample(which(ts$class==2),20),])
scorer <- randomForest(class~., data=tr, ntree=500)
```

```
scores <- cbind(predict(scorer, validation, type = c("prob")), validation$class)
test.scores <- predict(scorer, ts_sample, type = c("prob"))
MKS(p.score = scores[scores[,3]==1,1], n.score = scores[scores[,3]==2,1],
test = test.scores)
```

MS *Median Sweep*

Description

It quantifies events based on testing scores, applying Median Sweep (MS) method, according to Forman (2006).

Usage

```
MS(test, TprFpr)
```

Arguments

test	a numeric vector containing the score estimated for the positive class from each test set instance.
TprFpr	a data.frame of true positive (tpr) and false positive (fpr) rates estimated on training set, using the function getTPRandFPRbyThreshold().

Value

the class distribution in the test set.

References

Forman, G. (2006, August). Quantifying trends accurately despite classifier error and class imbalance. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 157-166).<doi.org/10.1145/1150402.1150423>.

Examples

```
library(randomForest)
library(caret)
cv <- createFolds(aeAegypti$class, 3)
tr <- aeAegypti[cv$Fold1,]
validation <- aeAegypti[cv$Fold2,]
ts <- aeAegypti[cv$Fold3,]

# -- Getting a sample from ts with 80 positive and 20 negative instances --
ts_sample <- rbind(ts[sample(which(ts$class==1),80),],
                   ts[sample(which(ts$class==2),20),])
scorer <- randomForest(class~., data=tr, ntree=500)
scores <- cbind(predict(scorer, validation, type = c("prob")), validation$class)
TprFpr <- getTPRandFPRbyThreshold(scores)
```

```
test.scores <- predict(scorer, ts_sample, type = c("prob"))
MS(test = test.scores[,1], TprFpr = TprFpr)
```

MS2 *Threshold selection method. Median Sweep*

Description

It quantifies events using a modified version of the MS method that considers only thresholds where the denominator (tpr-fpr) is greater than 0.25.

Usage

```
MS2(test, TprFpr)
```

Arguments

test	a numeric vector containing the score estimated for the positive class from each test set instance.
TprFpr	a data.frame of true positive (tpr) and false positive (fpr) rates estimated on training set, using the function getTPRandFPRbyThreshold().

Value

the class distribution in the test set.

References

Forman, G. (2006, August). Quantifying trends accurately despite classifier error and class imbalance. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 157-166).<doi.org/10.1145/1150402.1150423>.

Examples

```
library(randomForest)
library(caret)
cv <- createFolds(aeAegypti$class, 3)
tr <- aeAegypti[cv$Fold1,]
validation <- aeAegypti[cv$Fold2,]
ts <- aeAegypti[cv$Fold3,]

# -- Getting a sample from ts with 80 positive and 20 negative instances --
ts_sample <- rbind(ts[sample(which(ts$class==1),80),],
                   ts[sample(which(ts$class==2),20),])
scorer <- randomForest(class~., data=tr, ntree=500)
scores <- cbind(predict(scorer, validation, type = c("prob")), validation$class)
TprFpr <- getTPRandFPRbyThreshold(scores)
test.scores <- predict(scorer, ts_sample, type = c("prob"))
MS2(test = test.scores[,1], TprFpr = TprFpr)
```

PACC *Probabilistic Adjusted Classify and Count*

Description

It quantifies events based on testing scores, applying the Probabilistic Adjusted Classify and Count (PACC) method. This method is also called Scaled Probability Average (SPA).

Usage

```
PACC(test, TprFpr, thr=0.5)
```

Arguments

`test`	a numeric `vector` containing the score estimated for the positive class from each test set instance. (NOTE: It requires calibrated scores. See calibrate from **CORElearn**).
`TprFpr`	a `data.frame` of true positive (`tpr`) and false positive (`fpr`) rates estimated on training set, using the function `getTPRandFPRbyThreshold()`.
`thr`	threshold value according to the `tpr` and `fpr` were learned. Default is `0.5`.

Value

the class distribution in the test set.

References

Bella, A., Ferri, C., Hernández-Orallo, J., & Ramírez-Quintana, M. J. (2010). Quantification via probability estimators. In IEEE International Conference on Data Mining (pp. 737–742). Sidney.<doi.org/10.1109/ICDM.2010.75>.

Examples

```
library(randomForest)
library(caret)
cv <- createFolds(aeAegypti$class, 3)
tr <- aeAegypti[cv$Fold1,]
validation <- aeAegypti[cv$Fold2,]
ts <- aeAegypti[cv$Fold3,]

# -- Getting a sample from ts with 80 positive and 20 negative instances --
ts_sample <- rbind(ts[sample(which(ts$class==1),80),],
                   ts[sample(which(ts$class==2),20),])
scorer <- randomForest(class~., data=tr, ntree=500)
scores <- cbind(predict(scorer, validation, type = c("prob")), validation$class)
TprFpr <- getTPRandFPRbyThreshold(scores)
test.scores <- predict(scorer, ts_sample, type = c("prob"))[,1]
```

```
# -- PACC requires calibrated scores. Be aware of doing this before using PACC --
# -- You can make it using calibrate function from the CORElearn package --
if(requireNamespace("CORElearn")){
   cal_tr <- CORElearn::calibrate(as.factor(scores[,3]), scores[,1], class1=1,
   method="isoReg",assumeProbabilities=TRUE)
   test.scores <- CORElearn::applyCalibration(test.scores, cal_tr)
}
PACC(test = test.scores, TprFpr = TprFpr)
```

PCC *Probabilistic Classify and Count*

Description

It quantifies events based on testing scores, applying the Probabilistic Classify and Count (PCC) method.

Usage

```
PCC(test)
```

Arguments

test	a numeric vector containing the score estimated for the positive class from each test set instance. (NOTE: It requires calibrated scores. See calibrate from **CORElearn**).

Value

the class distribution in the test set.

References

Bella, A., Ferri, C., Hernández-Orallo, J., & Ramírez-Quintana, M. J. (2010). Quantification via probability estimators. In IEEE International Conference on Data Mining (pp. 737–742). Sidney.<doi.org/10.1109/ICDM.2010.75>.

Examples

```
library(randomForest)
library(caret)
cv <- createFolds(aeAegypti$class, 3)
tr <- aeAegypti[cv$Fold1,]
validation <- aeAegypti[cv$Fold2,]
ts <- aeAegypti[cv$Fold3,]

# -- Getting a sample from ts with 80 positive and 20 negative instances --
ts_sample <- rbind(ts[sample(which(ts$class==1),80),],
                   ts[sample(which(ts$class==2),20),])
```

```
scorer <- randomForest(class~., data=tr, ntree=500)
scores <- cbind(predict(scorer, validation, type = c("prob")), validation$class)
test.scores <- predict(scorer, ts_sample, type = c("prob"))[,1]

# -- PCC requires calibrated scores. Be aware of doing this before using PCC --
# -- You can make it using calibrate function from the CORElearn package --
if(requireNamespace("CORElearn")){
  cal_tr <- CORElearn::calibrate(as.factor(scores[,3]), scores[,1], class1=1,
  method="isoReg",assumeProbabilities=TRUE)
  test.scores <- CORElearn::applyCalibration(test.scores, cal_tr)
}
PCC(test=test.scores)
```

SMM *Sample Mean Matching*

Description

SMM is a member of the DyS framework that uses simple means scores to represent the score distribution for positive, negative, and unlabelled scores. Therefore, the class distribution is given by a closed-form equation.

Usage

```
SMM(p.score, n.score, test)
```

Arguments

`p.score`	a numeric `vector` of positive scores estimated either from a validation set or from a cross-validation method.
`n.score`	a numeric `vector` of negative scores estimated either from a validation set or from a cross-validation method.
`test`	a numeric `vector` containing the score estimated for the positive class from each test set instance.

References

Hassan, W., Maletzke, A., Batista, G. (2020). Accurately Quantifying a Billion Instances per Second. In IEEE International Conference on Data Science and Advanced Analytics (DSAA).

Examples

```
library(randomForest)
library(caret)
cv <- createFolds(aeAegypti$class, 3)
tr <- aeAegypti[cv$Fold1,]
validation <- aeAegypti[cv$Fold2,]
ts <- aeAegypti[cv$Fold3,]
```

```
# -- Getting a sample from ts with 80 positive and 20 negative instances --
ts_sample <- rbind(ts[sample(which(ts$class==1),80),],
                   ts[sample(which(ts$class==2),20),])
scorer <- randomForest(class~., data=tr, ntree=500)
scores <- cbind(predict(scorer, validation, type = c("prob")), validation$class)
test.scores <- predict(scorer, ts_sample, type = c("prob"))
SMM(p.score = scores[scores[,3]==1,1], n.score = scores[scores[,3]==2,1],
test = test.scores[,1])
```

SORD *Sample ORD Dissimilarity*

Description

It quantifies events based on testing scores applying the framework DyS with the Sample ORD Dissimilarity (SORD) proposed by Maletzke et al. (2019).

Usage

```
SORD(p.score, n.score, test)
```

Arguments

p.score	a numeric vector of positive scores estimated either from a validation set or from a cross-validation method.
n.score	a numeric vector of negative scores estimated either from a validation set or from a cross-validation method.
test	a numeric vector containing the score estimated for the positive class from each test set instance.

Value

the estimated class distribution.

References

Maletzke, A., Reis, D., Cherman, E., & Batista, G. (2019). DyS: a Framework for Mixture Models in Quantification. in Proceedings of the The Thirty-Third AAAI Conference on Artificial Intelligence, ser. AAAI'19, 2019.<doi.org/10.1609/aaai.v33i01.33014552>.

Examples

```
library(randomForest)
library(caret)
cv <- createFolds(aeAegypti$class, 3)
tr <- aeAegypti[cv$Fold1,]
validation <- aeAegypti[cv$Fold2,]
```

```
ts <- aeAegypti[cv$Fold3,]

# -- Getting a sample from ts with 80 positive and 20 negative instances --
ts_sample <- rbind(ts[sample(which(ts$class==1),80),],
                   ts[sample(which(ts$class==2),20),])
scorer <- randomForest(class~., data=tr, ntree=500)
scores <- cbind(predict(scorer, validation, type = c("prob")), validation$class)
test.scores <- predict(scorer, ts_sample, type = c("prob"))
SORD(p.score = scores[scores[,3]==1,1], n.score = scores[scores[,3]==2,1],
test = test.scores[,1])
```

`T50` *Threshold selection method*

Description

It quantifies events based on testing scores, applying T50 method proposed by Forman (2006). It sets the decision threshold of Binary Classifier where `tpr` = 50%.

Usage

```
T50(test, TprFpr)
```

Arguments

`test`	a numeric `vector` containing the score estimated for the positive class from each test set instance.
`TprFpr`	a `data.frame` of true positive (`tpr`) and false positive (`fpr`) rates estimated on training set, using the function `getTPRandFPRbyThreshold()`.

Value

the class distribution in the test set.

References

Forman, G. (2006, August). Quantifying trends accurately despite classifier error and class imbalance. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 157-166).<doi.org/10.1145/1150402.1150423>.

Examples

```
library(randomForest)
library(caret)
cv <- createFolds(aeAegypti$class, 3)
tr <- aeAegypti[cv$Fold1,]
validation <- aeAegypti[cv$Fold2,]
ts <- aeAegypti[cv$Fold3,]
# -- Getting a sample from ts with 80 positive and 20 negative instances --
```

```
ts_sample <- rbind(ts[sample(which(ts$class==1),80),],
                   ts[sample(which(ts$class==2),20),])
scorer <- randomForest(class~., data=tr, ntree=500)
scores <- cbind(predict(scorer, validation, type = c("prob")), validation$class)
TprFpr <- getTPRandFPRbyThreshold(scores)
test.scores <- predict(scorer, ts_sample, type = c("prob"))
T50(test=test.scores[,1], TprFpr=TprFpr)
```

`TernarySearch` *Implements the Ternary Search algorithm*

Description

Ternary search is used for finding the best alpha value in DyS framework.

Usage

```
TernarySearch(left, right, f, eps = 1e-04)
```

Arguments

`left`	a numeric value containing the left bound.
`right`	a numeric value containing the right bound.
`f`	unimodal function.
`eps`	stop criterium.

Value

the minimum of a unimodal function `f()` within `[left,right]`.

Author(s)

Andre Maletzke <andregustavom@gmail.com>

`X` *Threshold selection method*

Description

It quantifies events based on testing scores, applying the X method (Forman, 2006). Same as T50, but set the threshold where (`1 - tpr`) = `fpr`.

Usage

```
X(test, TprFpr)
```

Arguments

test	a numeric vector containing the score estimated for the positive class from each test set instance.
TprFpr	a data.frame of true positive (tpr) and false positive (fpr) rates estimated on training set, using the function getTPRandFPRbyThreshold().

Value

the class distribution in the test set.

References

Forman, G. (2006, August). Quantifying trends accurately despite classifier error and class imbalance. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 157-166).<doi.org/10.1145/1150402.1150423>.

Examples

```
library(randomForest)
library(caret)
cv <- createFolds(aeAegypti$class, 3)
tr <- aeAegypti[cv$Fold1,]
validation <- aeAegypti[cv$Fold2,]
ts <- aeAegypti[cv$Fold3,]

# -- Getting a sample from ts with 80 positive and 20 negative instances --
ts_sample <- rbind(ts[sample(which(ts$class==1),80),],
                   ts[sample(which(ts$class==2),20),])
scorer <- randomForest(class~., data=tr, ntree=500)
scores <- cbind(predict(scorer, validation, type = c("prob")), validation$class)
TprFpr <- getTPRandFPRbyThreshold(scores)
test.scores <- predict(scorer, ts_sample, type = c("prob"))
X(test=test.scores[,1], TprFpr=TprFpr)
```

Index