

# Package ‘conformalClassification’

December 17, 2017

**Type** Package

**Title** conformalClassification: A Conformal Prediction R Package for Classification

**Version** 0.1.0

**Author** Niharika Gauraha and Ola Spjuth

**Maintainer** Niharika <niharika.gauraha@farmbio.uu.se>

**Description** A Conformal Prediction R Package for Classification Problems

**Depends** randomForest, foreach, doParallel, caret

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 6.0.1.9000

## R topics documented:

conformalClassification . . . . .	1
CPCalibrationPlot . . . . .	2
CPEfficiency . . . . .	3
CPErrorRate . . . . .	4
CPObsFuzziness . . . . .	5
CPValidity . . . . .	6
ICPClassification . . . . .	7
parTCPClassification . . . . .	8
TCPClassification . . . . .	9

<b>Index</b>	<b>11</b>
--------------	-----------

---

conformalClassification

*A Conformal Prediction R Package for Classification*

---

## Description

The conformalClassification package implements Transductive Conformal Prediction (TCP) and Inductive Conformal Prediction (ICP) for classification problems.

## Details

Currently, the package is built upon random forests method, where voting of random forests for each class is considered as a conformity scores for each data point. Mainly the package generates conformal prediction errors (p-values) for classification problems, it also provides various diagnostic measures such as deviation from alidity, error rate, efficiency, observed fuzziness and calibration plots. In future releases, we plan to extend package to use other machine learning algorithms, (i.e. support vector machine) for model fitting.

---

CPCalibrationPlot	<i>Plots the calibration plot</i>
-------------------	-----------------------------------

---

## Description

Plots the calibration plot

## Usage

```
CPCalibrationPlot(pValues, testSet, color = "blue")
```

## Arguments

testSet	The test set
color	colour of the calibration line
matPValues	Matrix of p-values

## See Also

[CPEfficiency](#), [CPErrorRate](#), [CPValidity](#), [CPObsFuzziness](#).

## Examples

```
## load the library
library(mlbench)

## load the DNA dataset
data(DNA)
originalData <- DNA

## make sure first column is always the label and class labels are always 1, 2, ...
nrAttr = ncol(originalData) #no of attributes
tempColumn = originalData[, 1]
originalData[, 1] = originalData[, nrAttr]
originalData[, nrAttr] = tempColumn
originalData[, 1] = as.factor(originalData[, 1])
originalData[, 1] = as.numeric(originalData[, 1])

## partition the data into training and test set
result = createDataPartition(originalData[, 1], p = 0.8, list = FALSE)
trainingSet = originalData[result, ]
testSet = originalData[-result, ]

##ICP classification
pValues = ICPClassification(trainingSet, testSet)
calibrationPlot(pValues, testSet, "blue")
```

---

CPEfficiency	<i>Computes efficiency of a conformal predictor, which is defined as the ratio of predictions with more than one class over the size of the testset</i>
--------------	---

---

**Description**

Computes efficiency of a conformal predictor, which is defined as the ratio of predictions with more than one class over the size of the testset

**Usage**

```
CPEfficiency(matPValues, testLabels, sigfLevel = 0.05)
```

**Arguments**

matPValues	Matrix of p-values
testLabels	True labels for the test-set
sigfLevel	Significance level

**Value**

The efficiency

**See Also**

[CPCalibrationPlot](#), [CPErrorRate](#), [CPValidity](#), [CPObsFuzziness](#).

**Examples**

```
## load the library
library(mlbench)

## load the DNA dataset
data(DNA)
originalData <- DNA

## make sure first column is always the label and class labels are always 1, 2, ...
nrAttr = ncol(originalData) #no of attributes
tempColumn = originalData[, 1]
originalData[, 1] = originalData[, nrAttr]
originalData[, nrAttr] = tempColumn
originalData[, 1] = as.factor(originalData[, 1])
originalData[, 1] = as.numeric(originalData[, 1])

## partition the data into training and test set
result = createDataPartition(originalData[, 1], p = 0.8, list = FALSE)
trainingSet = originalData[result, ]
testSet = originalData[-result, ]

##ICP classification
pValues = ICPClassification(trainingSet, testSet)
testLabels = testSet[,1]
CPEfficiency(pValues, testLabels)
```

---

CPErrrorRate	<i>Computes error rate of a conformal predictor, which is defined as the ratio of predictions with missing true class lables over the size of the testset</i>
--------------	---

---

### Description

Computes error rate of a conformal predictor, which is defined as the ratio of predictions with missing true class lables over the size of the testset

### Usage

```
CPErrrorRate(matPValues, testLabels, sigfLevel = 0.05)
```

### Arguments

matPValues	Matrix of p-values
testLabels	True labels for the test-set
sigfLevel	Significance level

### Value

The error rate

### See Also

[CPalibrationPlot](#), [CPEfficiency](#), [CPValidity](#), [CPObsFuzziness](#).

### Examples

```
## load the library
library(mlbench)

## load the DNA dataset
data(DNA)
originalData <- DNA

## make sure first column is always the label and class labels are always 1, 2, ...
nrAttr = ncol(originalData) #no of attributes
tempColumn = originalData[, 1]
originalData[, 1] = originalData[, nrAttr]
originalData[, nrAttr] = tempColumn
originalData[, 1] = as.factor(originalData[, 1])
originalData[, 1] = as.numeric(originalData[, 1])

## partition the data into training and test set
result = createDataPartition(originalData[, 1], p = 0.8, list = FALSE)
trainingSet = originalData[result, ]
testSet = originalData[-result, ]

##ICP classification
pValues = ICPClassification(trainingSet, testSet)
testLabels = testSet[,1]
CPErrrorRate(pValues, testLabels)
```

---

CPObsFuzziness	<i>Computes observed fuzziness, which is defined as the sum of all p-values for the incorrect class labels.</i>
----------------	---

---

**Description**

Computes observed fuzziness, which is defined as the sum of all p-values for the incorrect class labels.

**Usage**

```
CPObsFuzziness(matPValues, testLabels)
```

**Arguments**

matPValues	Matrix of p-values
testLabels	True labels for the test-set

**Value**

The observed fuzziness

**See Also**

[CPCalibrationPlot](#), [CPEfficiency](#), [CPErrorRate](#), [CPValidity](#).

**Examples**

```
## load the library
library(mlbench)

## load the DNA dataset
data(DNA)
originalData <- DNA

## make sure first column is always the label and class labels are always 1, 2, ...
nrAttr = ncol(originalData) #no of attributes
tempColumn = originalData[, 1]
originalData[, 1] = originalData[, nrAttr]
originalData[, nrAttr] = tempColumn
originalData[, 1] = as.factor(originalData[, 1])
originalData[, 1] = as.numeric(originalData[, 1])

## partition the data into training and test set
result = createDataPartition(originalData[, 1], p = 0.8, list = FALSE)
trainingSet = originalData[result, ]
testSet = originalData[-result, ]

##ICP classification
pValues = ICPClassification(trainingSet, testSet)
testLabels = testSet[,1]
CPObsFuzziness(pValues, testLabels)
```

---

CPValidity	<i>Computes the deviation from exact validity as the Euclidean norm of the difference of the observed error and the expected error</i>
------------	--

---

### Description

Computes the deviation from exact validity as the Euclidean norm of the difference of the observed error and the expected error

### Usage

```
CPValidity(matPValues = NULL, testLabels = NULL)
```

### Arguments

matPValues	Matrix of p-values
testLabels	True labels for the test-set

### Value

The deviation from exact validity

### See Also

[CPCalibrationPlot](#), [CPEfficiency](#), [CPErrorRate](#), [CPObsFuzziness](#).

### Examples

```
## load the library
library(mlbench)

## load the DNA dataset
data(DNA)
originalData <- DNA

## make sure first column is always the label and class labels are always 1, 2, ...
nrAttr = ncol(originalData) #no of attributes
tempColumn = originalData[, 1]
originalData[, 1] = originalData[, nrAttr]
originalData[, nrAttr] = tempColumn
originalData[, 1] = as.factor(originalData[, 1])
originalData[, 1] = as.numeric(originalData[, 1])

## partition the data into training and test set
result = createDataPartition(originalData[, 1], p = 0.8, list = FALSE)
trainingSet = originalData[result, ]
testSet = originalData[-result, ]

##ICP classification
pValues = ICPClassification(trainingSet, testSet)
testLabels = testSet[,1]
CPValidity(pValues, testLabels)
```

---

ICPClassification	<i>Class-conditional Inductive conformal classifier for multi-class problems</i>
-------------------	--

---

**Description**

Class-conditional Inductive conformal classifier for multi-class problems

**Usage**

```
ICPClassification(trainingSet, testSet, ratioTrain = 0.7, method = "rf",  
  nrTrees = 100)
```

**Arguments**

trainingSet	Training set
testSet	Test set
ratioTrain	The ratio for proper training set
method	Method for modeling
nrTrees	Number of trees for RF

**Value**

The p-values

**See Also**

[TCPClassification](#), [parTCPClassification](#).

**Examples**

```
## load the library  
library(mlbench)  
  
## load the DNA dataset  
data(DNA)  
originalData <- DNA  
  
## make sure first column is always the label and class labels are always 1, 2, ...  
nrAttr = ncol(originalData) #no of attributes  
tempColumn = originalData[, 1]  
originalData[, 1] = originalData[, nrAttr]  
originalData[, nrAttr] = tempColumn  
originalData[, 1] = as.factor(originalData[, 1])  
originalData[, 1] = as.numeric(originalData[, 1])  
  
## partition the data into training and test set  
result = createDataPartition(originalData[, 1], p = 0.8, list = FALSE)  
trainingSet = originalData[result, ]  
testSet = originalData[-result, ]  
  
##ICP classification
```

```
pValues = ICPClassification(trainingSet, testSet)
#perfVlaues = pValues2PerfMetrics(pValues, testSet)
#print(perfVlaues)
#calibrationPlot(pValues, testSet, "blue")
```

---

parTCPClassification	<i>Class-conditional transductive conformal classifier for multi-class problems, paralld computations</i>
----------------------	---

---

## Description

Class-conditional transductive conformal classifier for multi-class problems, paralld computations

## Usage

```
parTCPClassification(trainSet, testSet, method = "rf", nrTrees = 100)
```

## Arguments

testSet	Test set
method	Method for modeling
nrTrees	Number of trees for RF
trainingSet	Training set

## Value

The p-values

## See Also

[TCPClassification](#). [ICPClassification](#).

## Examples

```
## load the library
library(mlbench)

## load the DNA dataset
data(DNA)
originalData <- DNA

## make sure first column is always the label and class labels are always 1, 2, ...
nrAttr = ncol(originalData) #no of attributes
tempColumn = originalData[, 1]
originalData[, 1] = originalData[, nrAttr]
originalData[, nrAttr] = tempColumn
originalData[, 1] = as.factor(originalData[, 1])
originalData[, 1] = as.numeric(originalData[, 1])

## partition the data into training and test set
result = createDataPartition(originalData[, 1], p = 0.8, list = FALSE)
trainingSet = originalData[result, ]
testSet = originalData[-result, ]
```



```
##ICP classification
pValues = parTCPClassification(trainingSet, testSet)
#perfVlaues = pValues2PerfMetrics(pValues, testSet)
#print(perfVlaues)
#calibrationPlot(pValues, testSet, "blue")
```

---

TCPClassification	<i>Class-conditional transductive conformal classifier for multi-class problems</i>
-------------------	---

---

## Description

Class-conditional transductive conformal classifier for multi-class problems

## Usage

```
TCPClassification(trainSet, testSet, method = "rf", nrTrees = 100)
```

## Arguments

testSet	Test set
method	Method for modeling
nrTrees	Number of trees for RF
trainingSet	Training set

## Value

The p-values

## See Also

[parTCPClassification](#), [ICPClassification](#).

## Examples

```
## load the library
library(mlbench)

## load the DNA dataset
data(DNA)
originalData <- DNA

## make sure first column is always the label and class labels are always 1, 2, ...
nrAttr = ncol(originalData) #no of attributes
tempColumn = originalData[, 1]
originalData[, 1] = originalData[, nrAttr]
originalData[, nrAttr] = tempColumn
originalData[, 1] = as.factor(originalData[, 1])
originalData[, 1] = as.numeric(originalData[, 1])

## partition the data into training and test set
result = createDataPartition(originalData[, 1], p = 0.8, list = FALSE)
```

```
trainingSet = originalData[result, ]
testSet = originalData[-result, ]

##reduce the size of the training set, because TCP is slow
result = createDataPartition(trainingSet[, 1], p=0.8, list=FALSE)
trainingSet = trainingSet[-result, ]

##TCP classification
pValues = TCPClassification(trainingSet, testSet)
#perfVlaues = pValues2PerfMetrics(pValues, testSet)
#print(perfVlaues)
#calibrationPlot(pValues, testSet, "blue")
```

# Index

conformalClassification, 1  
conformalClassification-package  
    (conformalClassification), 1  
CPalibrationPlot, 4  
CPCalibrationPlot, 2, 3, 5, 6  
CPEfficiency, 2, 3, 4–6  
CPErrorRate, 2, 3, 4, 5, 6  
CPObsFuzziness, 2–4, 5, 6  
CPValidity, 2–5, 6  
  
ICPClassification, 7, 8, 9  
  
parTCPClassification, 7, 8, 9  
  
TCPClassification, 7, 8, 9