

# Package ‘conformalClassification’

July 22, 2025

**Type** Package

**Title** Transductive and Inductive Conformal Predictions for Classification Problems

**Date** 2017-12-19

**Version** 1.0.0

**Author** Niharika Gauraha and Ola Spjuth

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

**Description** Implementation of transductive conformal prediction (see Vovk, 2013, <[doi:10.1007/978-3-642-41142-7\\_36](https://doi.org/10.1007/978-3-642-41142-7_36)>) and inductive conformal prediction (see Balasubramanian et al., 2014, ISBN:9780124017153) for classification problems.

**Depends** graphics, stats, randomForest, parallel, foreach, doParallel, mlbench

**License** GPL-3

**Encoding** UTF-8

**NeedsCompilation** no

**LazyData** true

**Repository** CRAN

**Date/Publication** 2017-12-22 18:29:54 UTC

## Contents

conformalClassification . . . . .	2
CPCalibrationPlot . . . . .	2
CPEfficiency . . . . .	3
CPErrorRate . . . . .	4
CPObsFuzziness . . . . .	6
CPValidity . . . . .	7
fitModel . . . . .	8
ICPClassification . . . . .	9
parTCPClassification . . . . .	10
TCPClassification . . . . .	11
tcpPValues . . . . .	12

---

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

*Plots the calibration plot*

---

### Description

Plots the calibration plot

### Usage

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

### Arguments

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

### See Also

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

**Examples**

```

## load the library
library(mlbench)
#library(caret)
library(conformalClassification)

## 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)
size = nrow(originalData)
result = sample(1:size, 0.8*size)
trainingSet = originalData[result, ]
testSet = originalData[-result, ]

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

```

---

CPEfficiency

*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*

---

**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](#), [CPErrrorRate](#), [CPValidity](#), [CPObsFuzziness](#).

**Examples**

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

## 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)
size = nrow(originalData)
result = sample(1:size, 0.8*size)

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

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

---

CPErrrorRate

*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*

---

**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**

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

**Examples**

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

## 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)
size = nrow(originalData)
result = sample(1:size, 0.8*size)

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](#), [CPErrrorRate](#), [CPValidity](#).

### Examples

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

## 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)
size = nrow(originalData)
result = sample(1:size, 0.8*size)
```

```

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](#), [CPErrrorRate](#), [CPObsFuzziness](#).

### Examples

```

## load the library
library(mlbench)
#library(caret)
library(conformalClassification)

## 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)
size = nrow(originalData)
result = sample(1:size, 0.8*size)

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

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

---

fitModel

*Fits the model and returns the fitted model*

---

### Description

Fits the model and returns the fitted model

### Usage

```
fitModel(trainingSet=NULL, method = "rf", nrTrees = 100)
```

### Arguments

trainingSet	The training set
method	Method for modeling
nrTrees	Number of trees for RF

### Value

The fitted model



---

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)  
#library(caret)  
library(conformalClassification)  
  
## 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)
size = nrow(originalData)
result = sample(1:size, 0.8*size)

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

##ICP classification
pValues = ICPClassification(trainingSet, testSet)
#perfVlaues = pValues2PerfMetrics(pValues, testSet)
#print(perfVlaues)
#CPCalibrationPlot(pValues, testSet, "blue")

```

---

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

---

### Description

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

### Usage

```
parTCPClassification(trainSet, testSet, method = "rf", nrTrees = 100, nrClusters = 12)
```

### Arguments

testSet	Test set
method	Method for modeling
nrTrees	Number of trees for RF
nrClusters	Number of clusters
trainSet	Training set

### Value

The p-values

### See Also

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

**Examples**

```

## load the library
#library(mlbench)
#library(caret)
#library(conformalClassification)

## 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)
#CPCalibrationPlot(pValues, testSet, "blue")
#not run

```

---

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
trainSet	Training set

**Value**

The p-values

**See Also**

[parTCPClassification](#). [ICPCClassification](#).

**Examples**

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

## 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)
#CPCalibrationPlot(pValues, testSet, "blue")
#not run
```

---

tcpPValues

*Fits the model and computes p-values*

---

**Description**

Fits the model and computes p-values

**Usage**

```
tcpPValues(augTrainSet, method = "rf", nrTrees = 100)
```

**Arguments**

augTrainSet	Augmented training set
method	Method for modeling
nrTrees	Number of trees for RF

**Value**

The p-values

# Index

conformalClassification, [2](#)  
conformalClassification-package  
    (conformalClassification), [2](#)  
CPCalibrationPlot, [2, 4–7](#)  
CPEfficiency, [2, 3, 5–7](#)  
CPErrrorRate, [2, 4, 4, 6, 7](#)  
CPObsFuzziness, [2, 4, 5, 6, 7](#)  
CPValidity, [2, 4–6, 7](#)  
  
fitModel, [8](#)  
  
ICPClassification, [9, 10, 12](#)  
  
parTCPClassification, [9, 10, 12](#)  
  
TCPClassification, [9, 10, 11](#)  
tcpPValues, [12](#)