Package ‘arc’

July 22, 2025

Title Association Rule Classification

Version 1.4.2

Date 2025-04-03

Maintainer Tomas Kliegr <kliegr@gmail.com>

Description Implements the Classification-based on
Association Rules (CBA) algorithm for association rule classification.
The package, also described in Hahsler et al. (2019) <doi:10.32614/RJ-2019-048>,
contains several convenience methods that allow to automatically
set CBA parameters (minimum confidence, minimum support) and it also natively
handles numeric attributes by integrating a pre-discretization step.
The rule generation phase is handled by the 'arules' package.
To further decrease the size of the CBA models produced by the 'arc' package, postprocessing by the
'qCBA' package is suggested.

Copyright The mdlp2.R script reuses parts of the code from the R
\`discretization\` package by HyunJi Kim (GPL license).

Depends R (>= 3.5.0), arules (>= 1.7-4), R.utils, discretization

License GPL-3

Encoding UTF-8

LazyData true

URL https://github.com/kliegr/arc

BugReports https://github.com/kliegr/arc/issues

Imports Matrix (>= 0.5-0), methods, datasets, utils

Suggests qCBA

RoxygenNote 7.3.2

NeedsCompilation no

Author Tomas Kliegr [aut, cre]

Repository CRAN

Date/Publication 2025-04-03 12:10:02 UTC

Contents

`applyCut` *Apply Cut Points to Vector*

Description

Applies cut points to vector.

Usage

```
applyCut(col, cuts, infinite_bounds, labels)
```

Arguments

`col` input vector with data.

`cuts` vector with cutpoints. There are several special values defined: `NULL` indicates that no discretization will be performed, but the value will be converted to factor `"All"` indicates all values will be merged into one.

`infinite_bounds` a logical indicating how the bounds on the extremes should look like. If set to `FALSE`, the leftmost/rightmost intervals will be bounded by the minimum and maximum in the respective column. If set to `TRUE`, the leftmost/rightmost intervals will be bounded by negative and positive infinity.

`labels` a logical indicating whether the bins of the discretized data should be represented by integer codes or as interval notation using (a;b] when set to TRUE.

Value

Vector with discretized data.

See Also

applyCuts

Examples

```
applyCut(datasets::iris[[1]], c(3,6), TRUE, TRUE)
```

---

applyCuts *Apply Cut Points to Data Frame*

Description

Applies cut points to input data frame.

Usage

```
applyCuts(df, cutp, infinite_bounds, labels)
```

Arguments

df	input data frame.
cutp	a list of vectors with cutpoints (for more information see applyCut).
infinite_bounds	a logical indicating how the bounds on the extremes should look like (for more information see applyCut)
labels	a logical indicating whether the bins of the discretized data should be represented by integer codes or as interval notation using (a;b] when set to TRUE.

Value

discretized data. If there was no discretization specified for some columns, these are returned as is.

See Also

applyCut

Examples

```
applyCuts(datasets::iris, list(c(5,6), c(2,3), "All", NULL, NULL), TRUE, TRUE)
```

cba *CBA Classifier*

Description

Learns a CBA rule set from supplied dataframe.

Usage

```
cba(datadf, classAtt, rulelearning_options = NULL, pruning_options = NULL)
```

Arguments

`datadf` a data frame with data.

`classAtt` the name of the class attribute.

`rulelearning_options`
: custom options for the rule learning algorithm overriding the default values. If not specified, the the topRules function is called and defaults specified there are used
`target_rule_count` (int) mining stops when the resulting rule set contains this number of rules;
`trim` (boolean) if set to TRUE and more than `target_rule_count` is discovered, only first `target_rule_count` rules will be returned.
`minsupp` (float) minimum support threshold
`minconf` (float) minimum confidence threshold
`minlen` (int) minimum length of rules, minlen=1 corresponds to rule with empty antecedent and one item in consequent. In general, rules with empty antecedent are not desirable for the subsequent pruning algorithm, therefore the value of this parameter should be set at least to value 2.
`maxlen` (int) maximum length of rules, should be equal or higher than minlen. A higher value may decrease the number of iterations to obtain target_rule_count rules, but it also increases the risk of initial combinatorial explosion and subsequent memory crash of the apriori rule learner.
`maxtime` (int) maximum number of seconds it should take ‘apriori‘ to obtain rules.
`find_conf_supp_thresholds` (boolean) whether to use automatic threshold detection or not.

`pruning_options`
: custom options for the pruning algorithm overriding the default values.

Value

Object of class CBARuleModel.

Examples

```
# Example using automatic threshold detection
cba(datasets::iris, "Species", rulelearning_options = list(target_rule_count = 50000))
# Example using manually set confidence and support thresholds
rm <- cba(datasets::iris, "Species", rulelearning_options = list(minsupp=0.01,
  minconf=0.5, minlen=1, maxlen=5, maxtime=1000, target_rule_count=50000, trim=TRUE,
  find_conf_supp_thresholds=FALSE))
inspect(rm@rules)
```

---

cbaCSV *Example CBA Workflow with CSV Input*

Description

Learns a CBA rule set and saves the resulting rule set back to csv.

Usage

```
cbaCSV(
  path,
  outpath = NULL,
  classAtt = NULL,
  idcolumn = NULL,
  rulelearning_options = NULL,
  pruning_options = NULL
)
```

Arguments

path	path to csv file with data.
outpath	path to write the rule set to.
classAtt	the name of the class attribute.
idcolumn	the name of the id column in the dataf ile.
rulelearning_options	custom options for the rule learning algorithm overriding the default values.
pruning_options	custom options for the pruning algorithm overriding the default values.

Value

Object of class CBARuleModel

Examples

```
# cbaCSV("path-to-.csv")
```

`cbaIris` *Test CBA Workflow on Iris Dataset*

Description

Test workflow on iris dataset: learns a cba classifier on one "train set" fold , and applies it to the second "test set" fold.

Usage

```
cbaIris()
```

Value

Accuracy.

`cbaIrisNumeric` *Test CBA Workflow on Iris Dataset with numeric target*

Description

Test workflow on iris dataset: learns a cba classifier on one "train set" fold, and applies it to the second "test set" fold.

Usage

```
cbaIrisNumeric()
```

Value

Accuracy.

`CBARuleModel-class` *CBARuleModel*

Description

This class represents a rule-based classifier.

Slots

`rules` an object of class rules from arules package

`cutp` list of cutpoints

`classAtt` name of the target class attribute

`attTypes` attribute types

CBARuleModelAccuracy — *Prediction Accuracy*

Description

Compares predictions with true labels and outputs accuracy.

Usage

```
CBARuleModelAccuracy(prediction, groundtruth)
```

Arguments

prediction	vector with predictions
groundtruth	vector with true labels

Value

Accuracy

cba_manual — *CBA Classifier from provided rules*

Description

Learns a CBA rule set from supplied rules

Usage

```
cba_manual(
  datadf_raw,
  rules,
  txns,
  rhs,
  classAtt,
  cutp,
  pruning_options = list(input_list_sorted_by_length = FALSE)
)
```

Arguments

datadf_raw	a data frame with raw data (numeric attributes are not discretized).
rules	Rules class instance output by the apriori package
txns	Transactions class instance passed to the arules method invocation. Transactions are created over discretized data frame - numeric values are replaced with intervals such as "(13;45]".
rhs	character vectors giving the labels of the items which can appear in the RHS ($rhs element of the APappearance class instance passed to the arules call)
classAtt	the name of the class attribute.
cutp	list of cutpoints used to discretize data (required for application of the model on continuous data)
pruning_options	custom options for the pruning algorithm overriding the default values.

Value

Object of class CBARuleModel.

Examples

```
data(humtemp)
data_raw<-humtemp
data_discr <- humtemp

#custom discretization
data_discr[,1]<-cut(humtemp[,1],breaks=seq(from=15,to=45,by=5))
data_discr[,2]<-cut(humtemp[,2],breaks=c(0,40,60,80,100))

#change interval syntax from (15,20] to (15;20], which is required by MARC
data_discr[,1]<-as.factor(unlist(lapply(data_discr[,1], function(x) {gsub(",", ";", x)})))
data_discr[,2]<-as.factor(unlist(lapply(data_discr[,2], function(x) {gsub(",", ";", x)})))
data_discr[,3] <- as.factor(humtemp[,3])

#mine rules
classAtt="Class"
appearance <- getAppearance(data_discr, classAtt)
txns_discr <- as(data_discr, "transactions")
rules <- apriori(txns_discr, parameter =
list(confidence = 0.5, support= 3/nrow(data_discr), minlen=1, maxlen=5), appearance=appearance)
inspect(rules)


rmCBA <- cba_manual(data_raw,  rules, txns_discr, appearance$rhs,
classAtt, cutp= list(), pruning_options=NULL)
inspect (rmCBA@rules)
prediction <- predict(rmCBA,data_discr,discretize=FALSE)
acc <- CBARuleModelAccuracy(prediction, data_discr[[classAtt]])
print(paste("Accuracy:",acc))
```

`discretizeUnsupervised` *Unsupervised Discretization*

Description

Discretizes provided numeric vector.

Usage

```
discretizeUnsupervised(
  data,
  labels = FALSE,
  infinite_bounds = FALSE,
  categories = 3,
  method = "cluster"
)
```

Arguments

`data`	input numeric vector.
`labels`	a logical indicating whether the bins of the discretized data should be represented by integer codes or as interval notation using (a;b] when set to TRUE.
`infinite_bounds`	a logical indicating how the bounds on the extremes should look like.
`categories`	number of categories (bins) to produce.
`method`	clustering method, one of "interval" (equal interval width), "frequency" (equal frequency), "cluster" (k-means clustering). See also documentation of the `discretize` function from the arules package.

Value

Discretized data. If there was no discretization specified for some columns, these are returned as is.

Examples

```
discretizeUnsupervised(datasets::iris[[1]])
```

`discrNumeric` *Discretize Numeric Columns In Data frame*

Description

Can discretize both predictor columns in data frame – using supervised algorithm MDLP (Fayyad & Irani, 1993) – and the target class – using unsupervised algorithm (k-Means). This R file contains fragments of code from the GPL-licensed R discretization package by HyunJi Kim.

Usage

```
discrNumeric(
  df,
  classAtt,
  min_distinct_values = 3,
  unsupervised_bins = 3,
  discretize_class = FALSE
)
```

Arguments

`df` a data frame with data.

`classAtt` name the class attribute in df

`min_distinct_values` the minimum number of unique values a column needs to have to be subject to supervised discretization.

`unsupervised_bins` number of target bins for discretizing the class attribute. Ignored when the class attribute is not numeric or when `discretize_class` is set to FALSE.

`discretize_class` logical value indicating whether the class attribute should be discretized. Ignored when the class attribute is not numeric.

Value

list with two slots: `$cutp` with cutpoints and `$Disc.data` with discretization results

References

Fayyad, U. M. and Irani, K. B. (1993). Multi-interval discretization of continuous-valued attributes for classification learning, Artificial intelligence 13, 1022–1027

Examples

```
discrNumeric(datasets::iris, "Species")
```

getAppearance — *Method that generates items for values in given data frame column.*

Description

Method that generates items for values in given data frame column.

Usage

```
getAppearance(df, classAtt)
```

Arguments

`df`	a data frame contain column `classAtt`.
`classAtt`	name of the column in `df` to generate items for.

Value

appearance object for mining classification rules

Examples

```
getAppearance(datasets::iris,"Species")
```

getConfVectorForROC — *Returns vector with confidences for the positive class (useful for ROC or AUC computation)*

Description

Methods for computing ROC curves require a vector of confidences of the positive class, while in CBA, the confidence returned by predict with outputProbabilies = TRUE returns confidence for the predicted class. This method converts the values to confidences for the positive class

Usage

```
getConfVectorForROC(confidences, predictedClass, positiveClass)
```

Arguments

`confidences`	Vector of confidences
`predictedClass`	Vector with predicted classes
`positiveClass`	Positive class (String)

Value

Vector of confidence values

Examples

```
predictedClass = c("setosa","virginica")
confidences = c(0.9,0.6)
baseClass="setosa"
getConfVectorForROC(confidences,predictedClass,baseClass)

# Further examples showing how ROC curve and AUC values can be computed
# using this function are available at project's GitHub homepage.
```

---

humtemp — *Comfort level based on temperature and humidity of the environment*

Description

A syntetic toy dataset. The variables are as follows:

Usage

```
data(humtemp)
```

Format

A data frame with 34 rows and 3 variables

Details

- Temperature.
- Humidity.
- Class. Comfort level

---

mdlp2 — *Supervised Discretization*

Description

Performs supervised discretization of numeric columns, except class, on the provided data frame. Uses the Minimum Description Length Principle algorithm (Fayyed and Irani, 1993) as implemented in the discretization package.

Usage

```
mdlp2(
  df,
  cl_index = NULL,
  handle_missing = FALSE,
  labels = FALSE,
  skip_nonnumeric = FALSE,
  infinite_bounds = FALSE,
  min_distinct_values = 3
)
```

Arguments

`df` input data frame.

`cl_index` index of the class variable. If not specified, the last column is used as the class variable.

`handle_missing` Setting to TRUE activates the following behaviour: if there are any missing observations in the column processed, the input for discretization is a subset of data containing this column and target with rows containing missing values excuded.

`labels` A logical indicating whether the bins of the discretized data should be represented by integer codes or as interval notation using (a;b] when set to TRUE.

`skip_nonnumeric` If set to TRUE, any non-numeric columns will be skipped.

`infinite_bounds` A logical indicating how the bounds on the extremes should look like.

`min_distinct_values` If a column contains less than specified number of distinct values, it is not discretized.

Value

Discretized data. If there were any non-numeric input columns they are returned as is. All returned columns except class are factors.

References

Fayyad, U. M. and Irani, K. B. (1993). Multi-interval discretization of continuous-valued attributes for classification learning, Artificial intelligence 13, 1022–1027

Examples

```
mdlp2(datasets::iris) #gives the same result as mdlp(datasets::iris) from discretize package
#uses Sepal.Length as target variable
mdlp2(df=datasets::iris, cl_index = 1,handle_missing = TRUE, labels = TRUE,
skip_nonnumeric = TRUE, infinite_bounds = TRUE, min_distinct_values = 30)
```

predict.CBARuleModel *Apply Rule Model*

Description

Method that matches rule model against test data.

Usage

```
## S3 method for class 'CBARuleModel'
predict(
  object,
  data,
  discretize = TRUE,
  outputFiringRuleIDs = FALSE,
  outputConfidenceScores = FALSE,
  confScoreType = "ordered",
  positiveClass = NULL,
  ...
)
```

Arguments

`object`	a CBARuleModel class instance
`data`	a data frame with data
`discretize`	boolean indicating whether the passed data should be discretized using information in the passed @cutp slot of the ruleModel argument.
`outputFiringRuleIDs`	if set to TRUE, instead of predictions, the function will return one-based IDs of rules used to classify each instance (one rule per instance).
`outputConfidenceScores`	if set to TRUE, instead of predictions, the function will return confidences of the firing rule
`confScoreType`	applicable only if ‘outputConfidenceScores=TRUE‘, possible values ‘ordered‘ for confidence computed only for training instances reaching this rule, or ‘global‘ for standard rule confidence computed from the complete training data
`positiveClass`	This setting is only used if ‘outputConfidenceScores=TRUE‘. It should be used only for binary problems. In this case, the confidence values are recalculated so that these are not confidence values of the predicted class (default behaviour of ‘outputConfidenceScores=TRUE‘) but rather confidence values associated with the class designated as positive
`...`	other arguments (currently not used)

Value

A vector with predictions.

See Also

cbaIris

Examples

```
set.seed(101)
allData <- datasets::iris[sample(nrow(datasets::iris)),]
trainFold <- allData[1:100,]
testFold <- allData[101:nrow(allData),]
#increase for more accurate results in longer time
target_rule_count <- 1000
classAtt <- "Species"
rm <- cba(trainFold, classAtt, list(target_rule_count = target_rule_count))
prediction <- predict(rm, testFold)
acc <- CBARuleModelAccuracy(prediction, testFold[[classAtt]])
message(acc)
# get rules responsible for each prediction
firingRuleIDs <- predict(rm, testFold, outputFiringRuleIDs=TRUE)
# show rule responsible for prediction of test instance no. 28
inspect(rm@rules[firingRuleIDs[28]])
# get prediction confidence (three different versions)
rm@rules[firingRuleIDs[28]]@quality$confidence
rm@rules[firingRuleIDs[28]]@quality$orderedConf
rm@rules[firingRuleIDs[28]]@quality$cumulativeConf
```

---

prune *Classifier Builder*

Description

An implementation of the CBA-CB M1 algorithm (Liu et al, 1998) adapted for R and arules package apriori implementation in place of CBA-RG.

Usage

```
prune(
  rules,
  txns,
  classitems,
  default_rule_pruning = TRUE,
  rule_window = 50000,
  greedy_pruning = FALSE,
  input_list_sorted_by_length = TRUE,
  debug = FALSE
)
```

Arguments

- `rules` — object of class rules from arules package
- `txns` — input object with transactions.
- `classitems` — a list of items to appear in the consequent (rhs) of the rules.
- `default_rule_pruning` — boolean indicating whether default pruning should be performed. If set to TRUE, default pruning is performed as in the CBA algorithm. If set to FALSE, default pruning is not performed i.e. all rules surviving data coverage pruning are kept. In either case, a default rule is added to the end of the classifier.
- `rule_window` — the number of rules to precompute for CBA data coverage pruning. The default value can be adjusted to decrease runtime.
- `greedy_pruning` — setting to TRUE activates early stopping condition: pruning will be stopped on first rule on which total error increases.
- `input_list_sorted_by_length` — indicates by default that the input rule list is sorted by antecedent length (as output by arules), if this param is set to false, the list will be resorted
- `debug` — output debug messages.

Value

Returns an object of class rules. Note that ‘rules@quality‘ slot has been extended with additional measures, specifically ‘orderedConf‘, ‘orderedSupp‘, and ‘cumulativeConf‘. The rules are output in the order in which they are assumed to be applied in classification. Only the first applicable rule is used to classify the instance. As a result, in addition to rule confidence – which is computed over the whole training dataset – it makes sense to define order-sensitive confidence, which is computed only from instances reaching the given rule as $a/(a + b)$, where a is the number of instances matching both the antecedent and consequent (available in slot ‘orderedSupp‘) and b is the number of instances matching the antecedent, but not matching the consequent of the given rule. The cumulative confidence is an experimental measure, which is computed as the accuracy of the rule list comprising the given rule and all higher priority rules (rules with lower index) with uncovered instances excluded from the computation.

References

Ma, Bing Liu Wynne Hsu Yiming. Integrating classification and association rule mining. Proceedings of the fourth international conference on knowledge discovery and data mining. 1998.

See Also

topRules

Examples

```
#Example 1
 txns <- as(discrNumeric(datasets::iris, "Species")$Disc.data,"transactions")
 appearance <- getAppearance(datasets::iris,"Species")
 rules <- apriori(txns, parameter = list(confidence = 0.5,
```

```
  support= 0.01, minlen= 2, maxlen= 4),appearance = appearance)
  prune(rules,txns, appearance$rhs)
  inspect(rules)

#Example 2
 utils::data(Adult) # this dataset comes with the arules package
 classitems <- c("income=small","income=large")
 rules <- apriori(Adult, parameter = list(supp = 0.3, conf = 0.5,
 target = "rules"), appearance=list(rhs=classitems, default="lhs"))
 # produces 25 rules
 rulesP <- prune(rules,Adult,classitems)
 rulesP@quality # inspect rule quality measured including the new additions
 # Rules after data coverage pruning: 8
 # Performing default rule pruning.
 # Final rule list size:  6
```

`topRules` *Rule Generation*

Description

A wrapper for the apriori method from the arules package that iteratively changes mining parameters until a desired number of rules is obtained, all options are exhausted or a preset time limit is reached. Within the arc package, this function serves as a replacement for the CBA Rule Generation algorithm (Liu et al, 1998) – without pessimistic pruning – with general apriori implementation provided by existing fast R package **arules**.

Usage

```
topRules(
  txns,
  appearance = list(),
  target_rule_count = 1000,
  init_support = 0,
  init_conf = 0.5,
  conf_step = 0.05,
  supp_step = 0.05,
  minlen = 2,
  init_maxlen = 3,
  iteration_timeout = 2,
  total_timeout = 100,
  max_iterations = 30,
  trim = TRUE,
  debug = FALSE
)
```

Arguments

`txns` input transactions.

`appearance` object named list or APappearance object (refer to arules package)

`target_rule_count`
: the main stopping criterion, mining stops when the resulting rule set contains this number of rules.

`init_support` initial support.

`init_conf` initial confidence.

`conf_step` confidence will be changed by steps defined by this parameter.

`supp_step` support will be changed by steps defined by this parameter.

`minlen` minimum length of rules, minlen=1 corresponds to rule with empty antecedent and one item in consequent. In general, rules with empty antecedent are not desirable for the subsequent pruning algorithm, therefore the value of this parameter should be set at least to value 2.

`init_maxlen` maximum length of rules, should be equal or higher than minlen. A higher value may decrease the number of iterations to obtain target_rule_count rules, but it also increases the risk of initial combinatorial explosion and subsequent memory crash of the apriori rule learner.

`iteration_timeout`
: maximum number of seconds it should take apriori to obtain rules with current configuration/

`total_timeout` maximum number of seconds the mining should take.

`max_iterations` maximum number of iterations.

`trim` if set to TRUE and more than `target_rule_count` is discovered, only first `target_rule_count` rules will be returned.

`debug` boolean indicating whether to output debug messages.

Value

Returns an object of class rules.

References

Ma, Bing Liu Wynne Hsu Yiming. Integrating classification and association rule mining. Proceedings of the fourth international conference on knowledge discovery and data mining. 1998.

See Also

prune

Examples

```
# Example 1
  utils::data(Adult)
  rules <- topRules(Adult, appearance = list(), target_rule_count = 100,
  init_support = 0.5,init_conf = 0.9, minlen = 1, init_maxlen = 10)

# Example 2
 rules <- topRules(as(discrNumeric(datasets::iris, "Species")$Disc.data,"transactions"),
  getAppearance(datasets::iris,"Species"))

# Example 3
  utils::data(datasets::iris)
  appearance <- list(rhs =  c("Species=setosa", "Species=versicolor",
   "Species=virginica"), default="lhs")
  data <- sapply(datasets::iris,as.factor)
  data <- data.frame(data, check.names=FALSE)
  txns <- as(data,"transactions")
  rules <- topRules(txns, appearance)
```

Index