Package: arulesCBA (via r-universe)

January 24, 2025

Version 1.2.7-1

Date 2024-xx-xx

Title Classification Based on Association Rules

Description Provides the infrastructure for association rule-based classification including the algorithms CBA, CMAR, CPAR, C4.5, FOIL, PART, PRM, RCAR, and RIPPER to build associative classifiers. Hahsler et al (2019) <doi:10.32614/RJ-2019-048>.

Maintainer Michael Hahsler <mhahsler@lyle.smu.edu>

Depends R (>= 4.0.0), Matrix (>= 1.4-0), arules (>= 1.7-4)

Imports methods, discretization (>= 1.0-1), glmnet (>= 3.0-0)

Suggests testthat, mlbench, rJava, RWeka

SystemRequirements Java (>= 8)

License GPL-3

URL https://github.com/mhahsler/arulesCBA

BugReports https://github.com/mhahsler/arulesCBA/issues

RoxygenNote 7.3.1

Roxygen list(markdown = TRUE)

Encoding UTF-8

Config/pak/sysreqs default-jdk

Repository https://mhahsler.r-universe.dev

RemoteUrl https://github.com/mhahsler/arulesCBA

RemoteRef HEAD

RemoteSha a8421bfcca593d5d78a9561368f93fd9739cb7a8

Contents

CBA *Classification Based on Association Rules Algorithm (CBA)*

Description

Build a classifier based on association rules using the ranking, pruning and classification strategy of the CBA algorithm by Liu, et al. (1998).

Usage

```
CBA(
  formula,
  data,
  pruning = "M1",
  parameter = NULL,
  control = NULL,
  balanceSupport = FALSE,
  disc.method = "mdlp",
  verbose = FALSE,
  ...
)

pruneCBA_M1(formula, rules, transactions, verbose = FALSE)

pruneCBA_M2(formula, rules, transactions, verbose = FALSE)
```

Arguments

`formula`	A symbolic description of the model to be fitted. Has to be of form `class ~ .` or `class ~ predictor1 + predictor2`.
`data`	arules::transactions containing the training data or a data.frame which. is automatically discretized and converted to transactions with `prepareTransactions()`.
`pruning`	Pruning strategy used: "M1" or "M2".

`parameter`, `control`
: Optional parameter and control lists for apriori.

`balanceSupport`
: balanceSupport parameter passed to `mineCARs()` function.

`disc.method`
: Discretization method used to discretize continuous variables if data is a data.frame (default: `"mdlp"`). See `discretizeDF.supervised()` for more supervised discretization methods.

`verbose`
: Show progress?

`...`
: For convenience, additional parameters are used to create the `parameter` control list for apriori (e.g., to specify the support and confidence thresholds).

`rules`, `transactions`
: prune a set of rules using a transaction set.

Details

Implementation the CBA algorithm with the M1 or M2 pruning strategy introduced by Liu, et al. (1998).

Candidate classification association rules (CARs) are mined with the APRIORI algorithm but minimum support is only checked for the LHS (rule coverage) and not the whole rule. Rules are ranked by confidence, support and size. Then either the M1 or M2 algorithm are used to perform database coverage pruning and default rule pruning.

Value

Returns an object of class CBA representing the trained classifier.

Author(s)

Ian Johnson and Michael Hahsler

References

Liu, B. Hsu, W. and Ma, Y (1998). Integrating Classification and Association Rule Mining. **KDD'98 Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining,** New York, 27-31 August. AAAI. pp. 80-86. https://dl.acm.org/doi/10.5555/3000292.3000305

See Also

Other classifiers: `CBA_helpers`, `CBA_ruleset()`, `FOIL()`, `LUCS_KDD_CBA`, `RCAR()`, `RWeka_CBA`

Examples

```
data("iris")

# 1. Learn a classifier using automatic default discretization
classifier <- CBA(Species ~ ., data = iris, supp = 0.05, conf = 0.9)
classifier

# inspect the rule base
```

```
inspect(classifier$rules)

# make predictions
predict(classifier, head(iris))
table(pred = predict(classifier, iris), true = iris$Species)


# 2. Learn classifier from transactions (and use verbose)
iris_trans <- prepareTransactions(Species ~ ., iris, disc.method = "mdlp")
iris_trans
classifier <- CBA(Species ~ ., data = iris_trans, supp = 0.05, conf = 0.9, verbose = TRUE)
classifier

# make predictions. Note: response extracts class information from transactions.
predict(classifier, head(iris_trans))
table(pred = predict(classifier, iris_trans), true = response(Species ~ ., iris_trans))
```

CBA_helpers *Helper Functions For Dealing with Classes*

Description

Helper functions to extract the response from transactions or rules, determine the class frequency, majority class, transaction coverage and the uncovered examples per class.

Usage

```
classes(formula, x)

response(formula, x)

classFrequency(formula, x, type = "relative")

majorityClass(formula, transactions)

transactionCoverage(transactions, rules)

uncoveredClassExamples(formula, transactions, rules)

uncoveredMajorityClass(formula, transactions, rules)
```

Arguments

formula	A symbolic description of the model to be fitted.
x, transactions	An object of class arules::transactions or arules::rules.
type	"relative" or "absolute"‘ to return proportions or absolute counts.
rules	A set of arules::rules.

Value

`response` returns the response label as a factor.

`classFrequency` returns the item frequency for each class label as a vector.

`majorityClass` returns the most frequent class label in the transactions.

Author(s)

Michael Hahsler

See Also

`arules::itemFrequency()`, arules::rules, arules::transactions.

Other classifiers: `CBA()`, `CBA_ruleset()`, `FOIL()`, `LUCS_KDD_CBA`, `RCAR()`, `RWeka_CBA`

Examples

```
data("iris")

iris.disc <- discretizeDF.supervised(Species ~ ., iris)
iris.trans <- as(iris.disc, "transactions")
inspect(head(iris.trans, n = 3))

# convert the class items back to a class label
response(Species ~ ., head(iris.trans, n = 3))

# Class labels
classes(Species ~ ., iris.trans)

# Class distribution. The iris dataset is perfectly balanced.
classFrequency(Species ~ ., iris.trans)

# Majority class
# (Note: since all class frequencies for iris are the same, the first one is returned)
majorityClass(Species ~ ., iris.trans)

# Use for CARs
cars <- mineCARs(Species ~ ., iris.trans, parameter = list(support = 0.3))

#' # Class labels
classes(Species ~ ., cars)

# Number of rules for each class
classFrequency(Species ~ ., cars, type = "absolute")

# conclusion (item in the RHS) of the rule as a class label
response(Species ~ ., cars)

# How many rules (using the first three rules) cover each transactions?
transactionCoverage(iris.trans, cars[1:3])

# Number of transactions per class not covered by the first three rules
```

```
uncoveredClassExamples(Species ~ ., iris.trans, cars[1:3])

# Majority class of the uncovered examples
uncoveredMajorityClass(Species ~ ., iris.trans, cars[1:3])
```

`CBA_ruleset` *Constructor for Objects for Classifiers Based on Association Rules*

Description

Objects for classifiers based on association rules have class `CBA`. A creator function `CBA_ruleset()` and several methods are provided.

Usage

```
CBA_ruleset(
  formula,
  rules,
  default,
  method = "first",
  weights = NULL,
  bias = NULL,
  model = NULL,
  discretization = NULL,
  description = "Custom rule set",
  ...
)
```

Arguments

`formula`	A symbolic description of the model to be fitted. Has to be of form `class ~ .`. The class is the variable name (part of the item label before =).
`rules`	A set of class association rules mined with `mineCARs()` or `arules::apriori()` (from **arules**).
`default`	Default class. If not specified then objects that are not matched by rules are classified as `NA`.
`method`	Classification method `"first"` found rule or `"majority"`.
`weights`	Rule weights for the majority voting method. Either a quality measure available in the classification rule set or a numeric vector of the same length are the classification rule set can be specified. If missing, then equal weights are used
`bias`	Class bias vector.
`model`	An optional list with model information (e.g., parameters).
`discretization`	A list with discretization information used by `predict()` to discretize data supplied as a `data.frame`.
`description`	Description field used when the classifier is printed.
`...`	Additional arguments added as list elements to the CBA object.

Details

CBA_ruleset() creates a new object of class CBA using the provides rules as the rule base. For method "first", the user needs to make sure that the rules are predictive and sorted from most to least predictive.

Value

A object of class CBA representing the trained classifier with fields:

formula	used formula.
rules	the classifier rule base.
default	default class label (uses partial matching against the class labels).
method	classification method.
weights	rule weights.
bias	class bias vector if available.
model	list with model description.
discretization	discretization information.
description	description in human readable form.

rules returns the rule base.

Author(s)

Michael Hahsler

See Also

mineCARs()

Other classifiers: CBA(), CBA_helpers, FOIL(), LUCS_KDD_CBA, RCAR(), RWeka_CBA

Other preparation: discretizeDF.supervised(), mineCARs(), prepareTransactions(), transactions2DF()

Examples

```
## Example 1: create a first-matching-rule classifier with non-redundant rules
##  sorted by confidence.
data("iris")

# discretize and create transactions
iris.disc <- discretizeDF.supervised(Species ~., iris)
trans <- as(iris.disc, "transactions")

# create rule base with CARs
cars <- mineCARs(Species ~ ., trans, parameter = list(support = .01, confidence = .8))

cars <- cars[!is.redundant(cars)]
cars <- sort(cars, by = "conf")

# create classifier and use the majority class as the default if no rule matches.
```

```
cl <- CBA_ruleset(Species ~ .,
  rules = cars,
  default = uncoveredMajorityClass(Species ~ ., trans, cars),
  method = "first")
cl

# look at the rule base
inspect(cl$rules)

# make predictions
prediction <- predict(cl, trans)
table(prediction, response(Species ~ ., trans))
accuracy(prediction, response(Species ~ ., trans))

# Example 2: use weighted majority voting.
cl <- CBA_ruleset(Species ~ .,
  rules = cars,
  default = uncoveredMajorityClass(Species ~ ., trans, cars),
  method = "majority", weights = "lift")
cl

prediction <- predict(cl, trans)
table(prediction, response(Species ~ ., trans))
accuracy(prediction, response(Species ~ ., trans))

## Example 3: Create a classifier with no rules that always predicts
##  the majority class. Note, we need cars for the structure and subset it
##  to leave no rules.
cl <- CBA_ruleset(Species ~ .,
  rules = cars[NULL],
  default = majorityClass(Species ~ ., trans))
cl

prediction <- predict(cl, trans)
table(prediction, response(Species ~ ., trans))
accuracy(prediction, response(Species ~ ., trans))
```

`discretizeDF.supervised` *Supervised Methods to Convert Continuous Variables into Categorical Variables*

Description

This function implements several supervised methods to convert continuous variables into a categorical variables (factor) suitable for association rule mining and building associative classifiers. A whole data.frame is discretized (i.e., all numeric columns are discretized).

Usage

```
discretizeDF.supervised(formula, data, method = "mdlp", dig.lab = 3, ...)
```

Arguments

formula	a formula object to specify the class variable for supervised discretization and the predictors to be discretized in the form class ~ . or class ~ predictor1 + predictor2.
data	a data.frame containing continuous variables to be discretized
method	discretization method. Available are: ‘"mdlp", "caim"‘, ‘"cacc"‘, ‘"ameva"‘, ‘"chi2"‘, ‘"chimerge"‘, ‘"extendedchi2"‘, and ‘"modchi2"‘.
dig.lab	integer; number of digits used to create labels.
...	Additional parameters are passed on to the implementation of the chosen discretization method.

Details

discretizeDF.supervised() only implements supervised discretization. See arules::discretizeDF() in package **arules** for unsupervised discretization.

Value

discretizeDF() returns a discretized data.frame. Discretized columns have an attribute "discretized:breaks" indicating the used breaks or and "discretized:method" giving the used method.

Author(s)

Michael Hahsler

See Also

Unsupervised discretization from **arules**: arules::discretize(), arules::discretizeDF().

Details about the available supervised discretization methods from **discretization**: discretization::mdlp, discretization::caim, discretization::cacc, discretization::ameva, discretization::chi2, discretization::chiM, discretization::extendChi2, discretization::modChi2.

Other preparation: CBA_ruleset(), mineCARs(), prepareTransactions(), transactions2DF()

Examples

```
data("iris")
summary(iris)

# supervised discretization using Species
iris.disc <- discretizeDF.supervised(Species ~ ., iris)
summary(iris.disc)

attributes(iris.disc$Sepal.Length)

# discretize the first few instances of iris using the same breaks as iris.disc
discretizeDF(head(iris), methods = iris.disc)

# only discretize predictors Sepal.Length and Petal.Length
```

```
iris.disc2 <- discretizeDF.supervised(Species ~ Sepal.Length + Petal.Length, iris)
head(iris.disc2)
```

FOIL *Use FOIL to learn a rule set for classification*

Description

Build a classifier rule base using FOIL (First Order Inductive Learner), a greedy algorithm that learns rules to distinguish positive from negative examples.

Usage

```
FOIL(
  formula,
  data,
  max_len = 3,
  min_gain = 0.7,
  best_k = 5,
  disc.method = "mdlp"
)
```

Arguments

formula	A symbolic description of the model to be fitted. Has to be of form `class ~ .` or `class ~ predictor1 + predictor2`.
data	A data.frame or arules::transactions containing the training data. Data frames are automatically discretized and converted to transactions with `prepareTransactions()`.
max_len	maximal length of the LHS of the created rules.
min_gain	minimal gain required to expand a rule.
best_k	use the average expected accuracy (laplace) of the best k rules per class for prediction.
disc.method	Discretization method used to discretize continuous variables if data is a data.frame (default: `"mdlp"`). See `discretizeDF.supervised()` for more supervised discretization methods.

Details

Implements FOIL (Quinlan and Cameron-Jones, 1995) to learn rules and then use them as a classifier following Xiaoxin and Han (2003).

For each class, we find the positive and negative examples and learn the rules using FOIL. Then the rules for all classes are combined and sorted by Laplace accuracy on the training data.

Following Xiaoxin and Han (2003), we classify new examples by

1. select all the rules whose bodies are satisfied by the example;
2. from the rules select the best k rules per class (highest expected Laplace accuracy);
3. average the expected Laplace accuracy per class and choose the class with the highest average.

Value

Returns an object of class CBA representing the trained classifier.

Author(s)

Michael Hahsler

References

Quinlan, J.R., Cameron-Jones, R.M. Induction of logic programs: FOIL and related systems. NGCO 13, 287-312 (1995). doi:10.1007/BF03037228

Yin, Xiaoxin and Jiawei Han. CPAR: Classification based on Predictive Association Rules, SDM, 2003. doi:10.1137/1.9781611972733.40

See Also

Other classifiers: CBA(), CBA_helpers, CBA_ruleset(), LUCS_KDD_CBA, RCAR(), RWeka_CBA

Examples

```
data("iris")

# learn a classifier using automatic default discretization
classifier <- FOIL(Species ~ ., data = iris)
classifier

# inspect the rule base
inspect(classifier$rules)

# make predictions for the first few instances of iris
predict(classifier, head(iris))
```

LUCS_KDD_CBA *Interface to the LUCS-KDD Implementations of CMAR, PRM and CPAR*

Description

Interface for the LUCS-KDD Software Library Java implementations of CMAR (Li, Han and Pei, 2001), PRM, and CPAR (Yin and Han, 2003). **Note:** The Java implementations is not part of **arulesCBA** and is only free for **non-commercial use**.

Usage

```
FOIL2(formula, data, best_k = 5, disc.method = "mdlp", verbose = FALSE)

CPAR(formula, data, best_k = 5, disc.method = "mdlp", verbose = FALSE)

PRM(formula, data, best_k = 5, disc.method = "mdlp", verbose = FALSE)

CMAR(
  formula,
  data,
  support = 0.1,
  confidence = 0.5,
  disc.method = "mdlp",
  verbose = FALSE
)
```

Arguments

formula	a symbolic description of the model to be fitted. Has to be of form `class ~ .` or `class ~ predictor1 + predictor2`.
data	A data.frame or arules::transactions containing the training data. Data frames are automatically discretized and converted to transactions with `prepareTransactions()`.
best_k	use average expected accuracy of the best k rules per class for prediction.
disc.method	Discretization method used to discretize continuous variables if data is a data.frame (default: `"mdlp"`). See `discretizeDF.supervised()` for more supervised discretization methods.
verbose	Show verbose output?
support, confidence	minimum support and minimum confidence thresholds for CMAR (range $[0, 1]$).

Details

Requirement: The code needs a **JDK (Java Software Development Kit) Version 1.8 (or higher)** installation. On some systems (Windows), you may need to set the `JAVA_HOME` environment variable so the system finds the compiler.

Memory: The memory for Java can be increased via R options. For example: `options(java.parameters = "-Xmx1024m")`

Note: The implementation does not expose the min. gain parameter for CPAR, PRM and FOIL2. It is fixed at 0.7 (the value used by Yin and Han, 2001). FOIL2 is an alternative Java implementation to the native implementation of FOIL already provided in the **arulesCBA**. FOIL exposes min. gain.

Value

Returns an object of class CBA representing the trained classifier.

References

Li W., Han, J. and Pei, J. CMAR: Accurate and Efficient Classification Based on Multiple Class-Association Rules, ICDM, 2001, pp. 369-376.

Yin, Xiaoxin and Jiawei Han. CPAR: Classification based on Predictive Association Rules, SDM, 2003. doi:10.1137/1.9781611972733.40

Frans Coenen et al. The LUCS-KDD Software Library, University of Liverpool, 2013.

See Also

Other classifiers: `CBA()`, `CBA_helpers`, `CBA_ruleset()`, `FOIL()`, `RCAR()`, `RWeka_CBA`

Examples

```
# make sure you have a Java SDK Version 1.4.0+ and not a headless installation.
system("java -version")

data("iris")

# build a classifier, inspect rules and make predictions
cl <- CMAR(Species ~ ., iris, support = .2, confidence = .8, verbose = TRUE)
cl

inspect(cl$rules)
predict(cl, head(iris))

cl <- CPAR(Species ~ ., iris)
cl

cl <- PRM(Species ~ ., iris)
cl

cl <- FOIL2(Species ~ ., iris)
cl
```

`Lymphography` *The Lymphography Domain Data Set (UCI)*

Description

This is lymphography domain obtained from the University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia. It was repeatedly used in the machine learning literature.

Format

A data frame with 147 observations on the following 19 variables.

`class` a factor with levels `normalfind` `metastases` `malignlymph` `fibrosis`

`lymphatics` a factor with levels `normal` `arched` `deformed` `displaced`

`blockofaffere` a factor with levels `no` `yes`

`bloflymphc` a factor with levels `no` `yes`

`bloflymphs` a factor with levels `no` `yes`

`bypass` a factor with levels `no` `yes`

`extravasates` a factor with levels `no` `yes`

`regenerationof` a factor with levels `no` `yes`

`earlyuptakein` a factor with levels `no` `yes`

`lymnodesdimin` a factor with levels `0` `1` `2` `3`

`lymnodesenlar` a factor with levels `1` `2` `3` `4`

`changesinlym` a factor with levels `bean` `oval` `round`

`defectinnode` a factor with levels `no` `lacunar` `lacmarginal` `laccentral`

`changesinnode` a factor with levels `no` `lacunar` `lacmargin` `laccentral`

`changesinstru` a factor with levels `no` `grainy` `droplike` `coarse` `diluted` `reticular` `stripped` `faint`

`specialforms` a factor with levels `no` `chalices` `vesicles`

`dislocationof` a factor with levels `no` `yes`

`exclusionofno` a factor with levels `no` `yes`

`noofnodesin` a factor with levels `0-9` `10-19` `20-29` `30-39` `40-49` `50-59` `60-69` `>=70`

Source

The data set was obtained from the UCI Machine Learning Repository at http://archive.ics.uci.edu/ml/datasets/Lymphography.

References

This lymphography domain was obtained from the University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia. Thanks go to M. Zwitter and M. Soklic for providing the data. Please include this citation if you plan to use this database.

Examples

```
data("Lymphography")

summary(Lymphography)
```

mineCARs *Mine Class Association Rules*

Description

Class Association Rules (CARs) are association rules that have only items with class values in the RHS as introduced for the CBA algorithm by Liu et al., 1998.

Usage

```
mineCARs(
  formula,
  transactions,
  parameter = NULL,
  control = NULL,
  balanceSupport = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

formula	A symbolic description of the model to be fitted.
transactions	An object of class arules::transactions containing the training data.
parameter, control	Optional parameter and control lists for arules::apriori().
balanceSupport	logical; if TRUE, class imbalance is counteracted by using class specific minimum support values. Alternatively, a support value for each class can be specified (see Details section).
verbose	logical; report progress?
...	For convenience, the mining parameters for arules::apriori() can be specified as .... Examples are the support and confidence thresholds, and the maxlen of rules.

Details

Class association rules (CARs) are of the form

$$P \Rightarrow c_i,$$

where the LHS P is a pattern (i.e., an itemset) and c_i is a single items representing the class label.

Mining parameters. Mining parameters for arules::apriori() can be either specified as a list (or object of arules::APparameter) as argument parameter or, for convenience, as arguments in .... *Note:* mineCARs() uses by default a minimum support of 0.1 (for the LHS of the rules via parameter originalSupport = FALSE), a minimum confidence of 0.5 and a maxlen (rule length including items in the LHS and RHS) of 5.

Balancing minimum support. Using a single minimum support threshold for a highly class imbalanced dataset will lead to the problem, that minority classes will only be presented in very few rules. To address this issue, `balanceSupport = TRUE` can be used to adjust minimum support for each class dependent on the prevalence of the class (i.e., the frequency of the c_i in the transactions) similar to the minimum class support suggested for CBA by Liu et al (2000) we use

$$minsupp_i = minsupp_t \frac{supp(c_i)}{max(supp(C))},$$

where $max(supp(C))$ is the support of the majority class. Therefore, the defined minimum support is used for the majority class and then minimum support is scaled down for classes which are less prevalent, giving them a chance to also produce a reasonable amount of rules. In addition, a named numerical vector with a support values for each class can be specified.

Value

Returns an object of class arules::rules.

Author(s)

Michael Hahsler

References

Liu, B. Hsu, W. and Ma, Y (1998). Integrating Classification and Association Rule Mining. *KDD'98 Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining,* New York, 27-31 August. AAAI. pp. 80-86.

Liu B., Ma Y., Wong C.K. (2000) Improving an Association Rule Based Classifier. In: Zighed D.A., Komorowski J., Zytkow J. (eds) *Principles of Data Mining and Knowledge Discovery. PKDD 2000. Lecture Notes in Computer Science*, vol 1910. Springer, Berlin, Heidelberg.

See Also

Other preparation: `CBA_ruleset()`, `discretizeDF.supervised()`, `prepareTransactions()`, `transactions2DF()`

Examples

```
data("iris")

# discretize and convert to transactions
iris.trans <- prepareTransactions(Species ~ ., iris)

# mine CARs with items for "Species" in the RHS.
# Note: mineCars uses a default a minimum coverage (lhs support) of 0.1, a
#       minimum confidence of .5 and maxlen of 5
cars <- mineCARs(Species ~ ., iris.trans)
inspect(head(cars))

# specify minimum support and confidence
cars <- mineCARs(Species ~ ., iris.trans,
  parameter = list(support = 0.3, confidence = 0.9, maxlen = 3))
```

```
inspect(head(cars))

# for convenience this can also be written without a list for parameter using ...
cars <- mineCARs(Species ~ ., iris.trans, support = 0.3, confidence = 0.9, maxlen = 3)

# restrict the predictors to items starting with "Sepal"
cars <- mineCARs(Species ~ Sepal.Length + Sepal.Width, iris.trans)
inspect(cars)

# using different support for each class
cars <- mineCARs(Species ~ ., iris.trans, balanceSupport = c(
  "Species=setosa" = 0.1,
  "Species=versicolor" = 0.5,
  "Species=virginica" = 0.01), confidence = 0.9)
cars

# balance support for class imbalance
data("Lymphography")
Lymphography_trans <- as(Lymphography, "transactions")

classFrequency(class ~ ., Lymphography_trans)

# mining does not produce CARs for the minority classes
cars <- mineCARs(class ~ ., Lymphography_trans, support = .3, maxlen = 3)
classFrequency(class ~ ., cars, type = "absolute")

# Balance support by reducing the minimum support for minority classes
cars <- mineCARs(class ~ ., Lymphography_trans, support = .3, maxlen = 3,
  balanceSupport = TRUE)
classFrequency(class ~ ., cars, type = "absolute")

# Mine CARs from regular transactions (a negative class item is automatically added)
data(Groceries)
cars <- mineCARs(`whole milk` ~ ., Groceries,
  balanceSupport = TRUE, support = 0.01, confidence = 0.8)
inspect(sort(cars, by = "lift"))
```

Mushroom *The Mushroom Data Set (UCI)*

Description

The Mushroom data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. It contains information about 8123 mushrooms. 4208 (51.8\ edible and 3916 (48.2\ features plus the class attribute (edible or not).

Format

A data frame with 8123 observations on the following 23 variables.

`Class` a factor with levels `edible` `poisonous`

`CapShape` a factor with levels `bell` `conical` `flat` `knobbed` `sunken` `convex`

`CapSurf` a factor with levels `fibrous` `grooves` `smooth` `scaly`

`CapColor` a factor with levels `buff` `cinnamon` `red` `gray` `brown` `pink` `green` `purple` `white` `yellow`

`Bruises` a factor with levels `no` `bruises`

`Odor` a factor with levels `almond` `creosote` `foul` `anise` `musty` `none` `pungent` `spicy` `fishy`

`GillAttached` a factor with levels `attached` `free`

`GillSpace` a factor with levels `close` `crowded`

`GillSize` a factor with levels `broad` `narrow`

`GillColor` a factor with levels `buff` `red` `gray` `chocolate` `black` `brown` `orange` `pink` `green` `purple` `white` `yellow`

`StalkShape` a factor with levels `enlarging` `tapering`

`StalkRoot` a factor with levels `bulbous` `club` `equal` `rooted`

`SurfaceAboveRing` a factor with levels `fibrous` `silky` `smooth` `scaly`

`SurfaceBelowRing` a factor with levels `fibrous` `silky` `smooth` `scaly`

`ColorAboveRing` a factor with levels `buff` `cinnamon` `red` `gray` `brown` `orange` `pink` `white` `yellow`

`ColorBelowRing` a factor with levels `buff` `cinnamon` `red` `gray` `brown` `orange` `pink` `white` `yellow`

`VeilType` a factor with levels `partial`

`VeilColor` a factor with levels `brown` `orange` `white` `yellow`

`RingNumber` a factor with levels `none` `one` `two`

`RingType` a factor with levels `evanescent` `flaring` `large` `none` `pendant`

`Spore` a factor with levels `buff` `chocolate` `black` `brown` `orange` `green` `purple` `white` `yellow`

`Population` a factor with levels `brown` `yellow`

`Habitat` a factor with levels `woods` `grasses` `leaves` `meadows` `paths` `urban` `waste`

Source

The data set was obtained from the UCI Machine Learning Repository at http://archive.ics.uci.edu/ml/datasets/Mushroom.

References

Alfred A. Knopf (1981). Mushroom records drawn from The Audubon Society Field Guide to North American Mushrooms. G. H. Lincoff (Pres.), New York.

Examples

```
data(Mushroom)

summary(Mushroom)
```

predict.CBA *Model Prediction for Classifiers Based on Association Rules*

Description

Predicts classes for new data using a CBA classifier.

Usage

```
## S3 method for class 'CBA'
predict(object, newdata, type = c("class", "score"), ...)

accuracy(pred, true)
```

Arguments

object	An object of class CBA.
newdata	A data.frame or arules::transactions containing rows of new entries to be classified.
type	Predict "class" labels. Some classifiers can also return "scores".
...	Additional arguments are ignored.
pred, true	two factors with the same level representing the predictions and the ground truth (e.g., obtained with response()).

Value

A factor vector with the classification result.

Author(s)

Michael Hahsler

Examples

```
data("iris")

train_id <- sample(seq_len(nrow(iris)), 130)
iris_train <- iris[train_id, ]
iris_test <- iris[-train_id, ]

cl <- CBA(Species ~., iris_train)
pr <- predict(cl, iris_test)
pr

accuracy(pr, response(Species ~., iris_test))
```

`prepareTransactions` *Prepare Data for Associative Classification*

Description

Converts data.frame into transactions suitable for classification based on association rules.

Usage

```
prepareTransactions(
  formula,
  data,
  disc.method = "mdlp",
  logical2factor = TRUE,
  match = NULL
)
```

Arguments

`formula`	the formula.
`data`	a data.frame with the data.
`disc.method`	Discretization method used to discretize continuous variables if data is a data.frame (default: `"mdlp"`). See `discretizeDF.supervised()` for more supervised discretization methods.
`logical2factor`	logical; if `data` is a data.frame, should logical columns be recoded as factor with TRUE/FALSE to generate positive and negative items?
`match`	typically `NULL`. Only used internally if data is a already a set of transactions.

Details

To convert a data.frame into items in a transaction dataset for classification, the following steps are performed:

1. All continuous features are discretized using class-based discretization (default is MDLP) and each range is represented as an item.
2. Factors are converted into items, one item for each level.
3. Each logical is converted into an item.
4. If the class variable is a logical, then a negative class item is added.

Steps 1-3 are skipped if `data` is already a arules::transactions object.

Value

An object of class arules::transactions from **arules** with an attribute called `"disc_info"` that contains information on the used discretization for each column.

Author(s)

Michael Hahsler

See Also

arules::transactions, `transactions2DF()`.

Other preparation: `CBA_ruleset()`, `discretizeDF.supervised()`, `mineCARs()`, `transactions2DF()`

Examples

```
# Perform discretization and convert to transactions
data("iris")
iris_trans <- prepareTransactions(Species ~ ., iris)

inspect(head(iris_trans))
itemInfo(iris_trans)

# A negative class item is added for regular transaction data. Here we get the
# items "canned beer=TRUE" and "canned beer=FALSE".
# Note: backticks are needed in formulas with item labels that contain
# a space or special character.
data("Groceries")
g2 <- prepareTransactions(`canned beer` ~ ., Groceries)

inspect(head(g2))
ii <- itemInfo(g2)
ii[ii[["variables"]] == "canned beer", ]
```

RCAR *Regularized Class Association Rules for Multi-class Problems (RCAR+)*

Description

Build a classifier based on association rules mined for an input dataset and weighted with LASSO regularized logistic regression following RCAR (Azmi, et al., 2019). RCAR+ extends RCAR from a binary classifier to a multi-label classifier and can use support-balanced CARs.

Usage

```
RCAR(
  formula,
  data,
  lambda = NULL,
  alpha = 1,
  glmnet.args = NULL,
  cv.glmnet.args = NULL,
  parameter = NULL,
```

```
  control = NULL,
  balanceSupport = FALSE,
  disc.method = "mdlp",
  verbose = FALSE,
  ...
)
```

Arguments

formula	A symbolic description of the model to be fitted. Has to be of form `class ~ .` or `class ~ predictor1 + predictor2`.
data	A data.frame or arules::transactions containing the training data. Data frames are automatically discretized and converted to transactions with `prepareTransactions()`.
lambda	The amount of weight given to regularization during the logistic regression learning process. If not specified (`NULL`) then cross-validation is used to determine the best value (see Details section).
alpha	The elastic net mixing parameter. `alpha = 1` is the lasso penalty (default RCAR), and `alpha = 0` the ridge penalty.
cv.glmnet.args, glmnet.args	A list of arguments passed on to `glmnet::cv.glmnet()` and `glmnet::glmnet()`, respectively. See Example section.
parameter, control	Optional parameter and control lists for `arules::apriori()`.
balanceSupport	balanceSupport parameter passed to `mineCARs()`.
disc.method	Discretization method for factorizing numeric input (default: `"mdlp"`). See `discretizeDF.supervised()` for more supervised discretization methods.
verbose	Report progress?
...	For convenience, additional parameters are used to create the `parameter` control list for `arules::apriori()` (e.g., to specify the support and confidence thresholds).

Details

RCAR+ extends RCAR from a binary classifier to a multi-label classifier using regularized multinomial logistic regression via **glmnet**.

In arulesCBA, the class variable is always represented by a set of items. For a binary classification problem, we use an item and its compliment (typically called `<item label>=TRUE` and `<item label>=FALSE`). For a multi-label classification problem we use one item for each possible class label (format `<class item>=<label>`). See `prepareTransactions()` for details.

RCAR+ first mines CARs to find itemsets (LHS of the CARs) that are related to the class items. Then, a transaction x lhs(CAR) coverage matrix X is created. The matrix contains a 1 if the LHS of the CAR applies to the transaction, and 0 otherwise. A regularized multinomial logistic model to predict the true class y for each transaction given X is fitted. Note that the RHS of the CARs are actually ignored in this process, so the algorithm effectively uses rules consisting of each LHS of a CAR paired with each class label. This is important to keep in mind when trying to interpret the rules used in the classifier.

If lambda for regularization is not specified during training (`lambda = NULL`) then cross-validation is used to determine the largest value of lambda such that the error is within 1 standard error of the minimum (see `glmnet::cv.glmnet()` for how to perform cross-validation in parallel).

For the final classifier, we only keep the rules that have a weight greater than 0 for at least one class label. The rules include as the weight the beta coefficients of the model.

Prediction for a new transaction is performed in two steps:

1. Translate the transaction into a 0-1 coverage vector indicating what class association rule's LHS covers the transaction.
2. Calculate the predicted label given the multinomial logistic regression model.

Value

Returns an object of class CBA representing the trained classifier with the additional field `model` containing a list with the following elements:

`reg_model`	them multinomial logistic regression model as an object of class glmnet::glmnet.
`cv`	only available if `lambda = NULL` was specified. Contains the results for the cross-validation used determine lambda. We use by default `lambda.1se` to determine lambda.
`all_rules`	the actual classifier only contains the rules with non-zero weights. This field contains all rules used to build the classifier, including the rules with a weight of zero. This is consistent with the model in `reg_model`.

Author(s)

Tyler Giallanza and Michael Hahsler

References

M. Azmi, G.C. Runger, and A. Berrado (2019). Interpretable regularized class association rules algorithm for classification in a categorical data space. *Information Sciences,* Volume 483, May 2019. Pages 313-331.

See Also

Other classifiers: `CBA()`, `CBA_helpers`, `CBA_ruleset()`, `FOIL()`, `LUCS_KDD_CBA`, `RWeka_CBA`

Examples

```
data("iris")

classifier <- RCAR(Species ~ ., iris)
classifier

# inspect the rule base sorted by the larges class weight
inspect(sort(classifier$rules, by = "weight"))

# make predictions for the first few instances of iris
```

```
predict(classifier, head(iris))
table(pred = predict(classifier, iris), true = iris$Species)

# plot the cross-validation curve as a function of lambda and add a
# red line at lambda.1se used to determine lambda.
plot(classifier$model$cv)
abline(v = log(classifier$model$cv$lambda.1se), col = "red")

# plot the coefficient profile plot (regularization path) for each class
# label. Note the line for the chosen lambda is only added to the last plot.
# You can manually add it to the others.
plot(classifier$model$reg_model, xvar = "lambda", label = TRUE)
abline(v = log(classifier$model$cv$lambda.1se), col = "red")

#' inspect rule 11 which has a large weight for class virginica
inspect(classifier$model$all_rules[11])
```

RWeka_CBA *CBA classifiers based on rule-based classifiers in RWeka*

Description

Provides CBA-type classifiers based on RIPPER (Cohen, 1995), C4.5 (Quinlan, 1993) and PART (Frank and Witten, 1998) using the implementation in Weka via RWeka (Hornik et al, 2009). These classifiers do not mine CARs, but directly create rules.

Usage

```
RIPPER_CBA(formula, data, control = NULL, disc.method = "mdlp")

PART_CBA(formula, data, control = NULL, disc.method = "mdlp")

C4.5_CBA(formula, data, control = NULL, disc.method = "mdlp")
```

Arguments

formula	A symbolic description of the model to be fitted. Has to be of form `class ~ .` or `class ~ predictor1 + predictor2`.
data	A data.frame or arules::transactions containing the training data. Data frames are automatically discretized and converted to transactions with `prepareTransactions()`.
control	algorithmic control options for R/Weka Rule learners (see Details Section).
disc.method	Discretization method used to discretize continuous variables if data is a data.frame (default: `"mdlp"`). See `discretizeDF.supervised()` for more supervised discretization methods.

Details

You need to install package **RWeka** to use these classifiers.

See R/Weka functions `RWeka::JRip()` (RIPPER), `RWeka::J48()` (C4.5 rules), `RWeka::PART()` for algorithm details and how control options can be passed on via `control`. An example is given in the Examples Section below.

Memory for **RWeka** can be increased using the R options (e.g., `options(java.parameters = "-Xmx1024m")`) before **RWeka** or **rJava** is loaded or any RWeka-based classifier in this package is used.

Value

Returns an object of class CBA representing the trained classifier.

Author(s)

Michael Hahsler

References

W. W. Cohen (1995). Fast effective rule induction. In A. Prieditis and S. Russell (eds.), Proceedings of the 12th International Conference on Machine Learning, pages 115-123. Morgan Kaufmann. ISBN 1-55860-377-8.

E. Frank and I. H. Witten (1998). Generating accurate rule sets without global optimization. In J. Shavlik (ed.), Machine Learning: Proceedings of the Fifteenth International Conference. Morgan Kaufmann Publishers: San Francisco, CA.

R. Quinlan (1993). C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo, CA.

Hornik K, Buchta C, Zeileis A (2009). "Open-Source Machine Learning: R Meets Weka." *Computational Statistics,* 24(2), 225-232. doi:10.1007/s0018000801197

See Also

Other classifiers: `CBA()`, `CBA_helpers`, `CBA_ruleset()`, `FOIL()`, `LUCS_KDD_CBA`, `RCAR()`

Examples

```
# rJava and RWeka need to be installed

## Not run:
data("iris")

# learn a classifier using automatic default discretization
classifier <- RIPPER_CBA(Species ~ ., data = iris)
classifier

# inspect the rule base
inspect(classifier$rules)
```

```
# make predictions for the first few instances of iris
predict(classifier, head(iris))

table(predict(classifier, iris), iris$Species)

# C4.5
classifier <- C4.5_CBA(Species ~ ., iris)
inspect(classifier$rules)

# To use algorithmic options (here for PART), you need to load RWeka
library(RWeka)

# control options can be found using the Weka Option Wizard (WOW)
WOW(PART)

# build PART with control option U (Generate unpruned decision list) set to TRUE
classifier <- PART_CBA(Species ~ ., data = iris, control = RWeka::Weka_control(U = TRUE))
classifier
inspect(classifier$rules)
predict(classifier, head(iris))

## End(Not run)
```

`transactions2DF` *Convert Transactions to a Data.Frame*

Description

Convert transactions back into data.frames by combining the items for the same variable into a single column.

Usage

```
transactions2DF(transactions, itemLabels = FALSE)
```

Arguments

`transactions`	an object of class arules::transactions.
`itemLabels`	logical; use the complete item labels (variable=level) as the levels in the data.frame? By default, only the levels are used.

Value

Returns a data.frame.

Author(s)

Michael Hahsler

See Also

Other preparation: CBA_ruleset(), discretizeDF.supervised(), mineCARs(), prepareTransactions()

Examples

```
data("iris")
iris_trans <- prepareTransactions(Species ~ ., iris)
iris_trans

# standard conversion
iris_df <- transactions2DF(iris_trans)
head(iris_df)

# use item labels in the data.frame
iris_df2 <- transactions2DF(iris_trans, itemLabels = TRUE)
head(iris_df2)

# Conversion of transactions without variables in itemInfo
data("Groceries")
head(transactions2DF(Groceries), 2)

# Conversion of transactions prepared for classification
g2 <- prepareTransactions(`shopping bags` ~ ., Groceries)
head(transactions2DF(g2), 2)
```

Index