Package ‘arulesCBA’

Version 1.2.1
Date 2021-11-20
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.
Maintainer Michael Hahsler <mhahsler@lyle.smu.edu>
Depends R (>= 3.5.0), Matrix (>= 1.2-0), arules (>= 1.6-5)
Imports methods, discretization (>= 1.0-1), glmnet (>= 3.0-0)
Suggests testthat, mlbench, rJava, RWeka
License GPL-3
URL https://github.com/ianjohnson/arulesCBA
BugReports https://github.com/ianjohnson/arulesCBA
RoxygenNote 7.1.2
Encoding UTF-8
NeedsCompilation yes
Author Michael Hahsler [aut, cre, cph],
            Ian Johnson [aut, cph],
            Tyler Giallanza [ctb]
Repository CRAN
Date/Publication 2021-11-20 22:30:15 UTC

R topics documented:

CBA ................................................................. 2
CBA_helpers ...................................................... 4
CBA_ruleset ....................................................... 6
discretizeDF:supervised ....................................... 8
FOIL .......................................................... 10
LUCS_KDD_CBA .............................................. 11
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 A data.frame or a transaction set containing the training data. Data frames are automatically discretized and converted to transactions.
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 \code{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 \code{CBA.object} representing the trained classifier.

Author(s)

Ian Johnson and Michael Hahsler

References


See Also

\code{CBA.object}, \code{mineCARs}.

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(rules(classifier))

# 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**

response(formula, x)
classFrequency(formula, x, type = "relative")
majorityClass(formula, transactions)
transactionCoverage(transactions, rules)
uncoveredClassExamples(formula, transactions, rules)
uncoveredMajorityClass(formula, transactions, rules)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>formula</td>
<td>A symbolic description of the model to be fitted.</td>
</tr>
<tr>
<td>x, transactions</td>
<td>An object of class transactions or rules.</td>
</tr>
<tr>
<td>type</td>
<td>&quot;relative&quot; or &quot;absolute&quot; to return proportions or absolute counts.</td>
</tr>
<tr>
<td>rules</td>
<td>A set of rules.</td>
</tr>
</tbody>
</table>
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

itemFrequency, rules, transactions.

Examples

data("iris")

iris.disc <- discretizeDF.supervised(Species ~ ., iris)
iris.trans <- as(iris.disc, "transactions")
inspect(head(iris.trans, n = 2))

# convert the class items back to a class label
response(Species ~ ., head(iris.trans, n = 2))

# 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))

# 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 = NA,
  method = "first",
  weights = NULL,
  bias = NULL,
  model = NULL,
  discretization = NULL,
  description = "Custom rule set",
  ...
)

rules(x)

## S3 method for class 'CBA'
rules(x)

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

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 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 method majority. Either a quality measure available in rules or a numeric vector of the same length are rules can be specified. If missing, then equal weights are used

bias Class bias vector.

model An optional list with model information (e.g., parameters).
**CBA_ruleset**

- **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.
- **x, object**: An object of class CBA.
- **newdata**: A data.frame or transactions containing rows of new entries to be classified.
- **type**: Predict "class" labels. Some classifiers can also return "scores".

**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**

`CBA_ruleset()` returns an object of class CBA representing the trained classifier with fields:

- **formula**: used formula.
- **rules**: the classifier rule base.
- **default**: default class label or NA.
- **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.

`predict` returns predicted labels for `newdata`.
`rules` returns the rule base.

**Author(s)**

Michael Hahsler

**See Also**

`CBA, mineCARs, apriori, rules, transactions`. 
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 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**: discretize, discretizeDF.

Details about the available supervised discretization methods from **discretization**: mdp, caim, cacc, ameva, chi2, chimerge, extendedchi2, modchi2.

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 a transaction set containing the training data. Data frames are automatically discretized and converted to transactions.
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.object` representing the trained classifier.

Author(s)

Michael Hahsler

References


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

See Also

`CBA.object`.

Examples

data("iris")

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

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

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

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 not covered by the packages license. It will be downloaded and compiled separately. It is available free of charge 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, best_k = 5, disc.method = "mdlp", verbose = FALSE)

install_LUCS_KDD_CPAR(force = FALSE, source = "https://cgi.csc.liv.ac.uk/~frans/KDD/Software/FOIL_PRM_CPAR/foilPrmCpar.tgz")

install_LUCS_KDD_CMAR(force = FALSE, source = "https://cgi.csc.liv.ac.uk/~frans/KDD/Software/CMAR/cmar.tgz")

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 a transaction set containing the training data. Data frames are automatically discretized and converted to transactions.

best_k use 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.

verbose Show verbose output?

support, confidence minimum support and minimum confidence thresholds for CMAR (range [0, 1]).

force logical; force redownload, rebuilding and reinstallation?

source source for the code. A local file can be specified as a URI starting with file:// (see download.file).
Details

Installation: The LUCS-KDD code is not part of the package and has to be downloaded, compiled and installed using `install_LUCS_KDD_CMAR()` and `install_LUCS_KDD_CPAR()`. You need a complete Java JDK installation including the javac compiler. 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.object` 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, https://cgi.csc.liv.ac.uk/~frans/KDD/Software/

Examples

```r
## Not run:
data("iris")

# install and compile CMAR
install_LUCS_KDD_CMAR()

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

# install CPAR (also installs PRM and FOIL2)
install_LUCS_KDD_CPAR()

cl <- CPAR(Species ~ ., iris)
cl
cl <- PRM(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 transactions containing the training data.

parameter, control Optional parameter and control lists for the apriori algorithm.

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 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 item representing the class label.

Mining parameters. Mining parameters for \texttt{apriori} can be either specified as a list (or object of \texttt{APparameter}) as argument \texttt{parameter} or, for convenience, as arguments in . . . \texttt{Note: mineCARs} uses by default a minimum support of 0.1 (for the LHS of the rules via parameter \texttt{originalSupport} = \texttt{FALSE}), a minimum confidence of 0.5 and a \texttt{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, \texttt{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

\[ \text{minsupp}_i = \text{minsupp}_t \frac{\text{supp}(c_i)}{\max(\text{supp}(C))}, \]

where \( \max(\text{supp}(C)) \) 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 \texttt{rules}.

Author(s)

Michael Hahsler

References


See Also

\texttt{apriori, APparameter, rules, transactions}. 
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)

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"))
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%) are edible and 3916 (48.2%) are poisonous. The data contains 22 nominal 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


Examples

```r
data(Mushroom)
summary(Mushroom)
```

---

**prepareTransactions**  
**Prepare Data for Associative Classification**

**Description**

Data in a data.frame are discretized using class-based discretization and converted into transactions. For transaction data that was not created from a data.frame, a negative class item is added to create data for a binary classifier.

**Usage**

```r
prepareTransactions(formula, data, disc.method = "mdlp", 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.
- `match`: typically NULL. Only used internally if data is a already a set of transactions.

**Value**

An object of class `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

transactions, transactions2DF.

Examples

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

# A negative class item is added for regular transaction data (here "!canned beer")
# Note: backticks are needed in formulas with item labels that contain a space.
data("Groceries")
g2 <- prepareTransactions("canned beer" ~ ., Groceries)
inspect(head(g2))
```

---

**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-class classifier and can use support-balanced CARs.

**Usage**

```r
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 containing the training data.
- **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 `cv.glmnet` and `glmnet`, respectively. See Example section.
- **parameter, control**: Optional parameter and control lists for apriori.
- **balanceSupport**: balanceSupport parameter passed to `mineCARs` function.
- **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 apriori (e.g., to specify the support and confidence thresholds).

Details

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

If `lambda` is not specified (NULL) then cross-validation with the largest value of lambda such that error is within 1 standard error of the minimum is used to determine the best value (see `cv.glmnet`). See `cv.glmnet` for performing cross-validation in parallel.

Value

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

- **all.rules**: all rules used to build the classifier, including the rules with a weight of zero.
- **reg_model**: the multinomial logistic regression model as an object of class `glmnet`.
- **cv**: contains the results for the cross-validation used determine lambda.

Author(s)

Tyler Giallanza and Michael Hahsler

References

See Also

CBA.object, mineCARs, glmnet and cv.glmnet.

Examples

```r
data("iris")

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

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

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

# inspecting the regression model, plot the regularization path, and
# plot the cross-validation results to determine lambda
str(classifier$model$reg_model)
plot(classifier$model$reg_model)
plot(classifier$model$cv)

# show progress report and use 5 instead of the default 10 cross-validation folds.
classifier <- RCAR(Species~., iris, cv.glmnet.args = list(nfolds = 5), verbose = TRUE)
```

---

**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).

**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 a transaction set containing the training data. Data frames are automatically discretized and converted to transactions.
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 JRip (RIPPER), J48 (C4.5 rules) 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-basedclassifier in this package is used.

Value
Returns an object of class CBA.object representing the trained classifier.

Author(s)
Michael Hahsler

References

See Also
JRip (RIPPER), PART, CBA.object.

Examples

# You need to install rJava and RWeka

## Not run:
data("iris")

# learn a classifier using automatic default discretization
classifier <- RIPPER_CBA(Species ~ ., data = iris)
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 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.
transactions2DF

Author(s)
Michael Hahsler

See Also
transactions.

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

* datasets
  Lymphography, 14
  Mushroom, 18

* manip
  discretizeDF.supervised, 8

ameva, 9
APparameter, 16
apriori, 7, 15, 16

C4.5_CBA (RWeka_CBA), 22
cacc, 9
caim, 9
CBA, 2, 7
cba (CBA), 2
CBA.object, 3, 11, 13, 22, 23
CBA.object (CBA_ruleset), 6
CBA_helpers, 4
CBA_ruleset, 6
chi2, 9
chiM, 9
classFrequency (CBA_helpers), 4
CMAR (LUCS_KDD_CBA), 11
CPAR (LUCS_KDD_CBA), 11
cv.glmnet, 21, 22
discretize, 9
discretize (discretizeDF.supervised), 8
discretizeDF, 9
discretizeDF (discretizeDF.supervised), 8
discretizeDF.supervised, 3, 8, 10, 12, 19, 21, 23
download.file, 12
extendChi2, 9

FOIL, 10, 13
foil (FOIL), 10
FOIL2 (LUCS_KDD_CBA), 11

glmnet, 21, 22
install_LUCS_KDD_CMAR (LUCS_KDD_CBA), 11
install_LUCS_KDD_CPAR (LUCS_KDD_CBA), 11
itemFrequency, 5

J48, 23
JRip, 23

LUCS_KDD_CBA, 11
Lymphography, 14

majorityClass (CBA_helpers), 4
mdlp, 9
mineCARs, 2, 3, 7, 15, 21, 22
modChi2, 9
Mushroom, 18

PART, 23
PART_CBA (RWeka_CBA), 22
predict.CBA (CBA_ruleset), 6
prepareTransactions, 19
print.CBA (CBA_ruleset), 6
PRM (LUCS_KDD_CBA), 11
pruneCBA_M1 (CBA), 2
pruneCBA_M2 (CBA), 2

RCAR, 20
rcar (RCAR), 20
response (CBA_helpers), 4
RIPPER_CBA (RWeka_CBA), 22
rules, 4, 5, 7, 16
rules (CBA_ruleset), 6
RWeka_CBA, 22

transactionCoverage (CBA_helpers), 4
transactions, 4, 5, 7, 15, 16, 19, 20, 25
transactions2DF, 20, 24

uncoveredClassExamples (CBA_helpers), 4
uncoveredMajorityClass (CBA_helpers), 4