Package: RWeka (via r-universe)

October 5, 2024

Version 0.4-46

Title R/Weka Interface

Description An R interface to Weka (Version 3.9.3). Weka is a collection of machine learning algorithms for data mining tasks written in Java, containing tools for data pre-processing, classification, regression, clustering, association rules, and visualization. Package 'RWeka' contains the interface code, the Weka jar is in a separate package 'RWekajars'. For more information on Weka see <https://www.cs.waikato.ac.nz/ml/weka/>.

Depends R (>= 2.6.0)

Imports RWekajars (>= 3.9.3-1), rJava (>= 0.6-3), graphics, stats, utils, grid

Suggests partykit (>= 0.8.0), mlbench, e1071

SystemRequirements Java (>= 8)

License GPL-2

NeedsCompilation no

Author Kurt Hornik [aut, cre] (<https://orcid.org/0000-0003-4198-9911>), Christian Buchta [ctb], Torsten Hothorn [ctb], Alexandros Karatzoglou [ctb], David Meyer [ctb], Achim Zeileis [ctb] (<https://orcid.org/0000-0003-0918-3766>)

Maintainer Kurt Hornik <Kurt.Hornik@R-project.org>

Repository CRAN

Date/Publication 2023-03-07 14:18:59 UTC

Contents

`dot` *Create DOT Representations*

Description

Write a DOT language representation of an object for processing via Graphviz.

Usage

```
write_to_dot(x, con = stdout(), ...)
## S3 method for class 'Weka_classifier'
write_to_dot(x, con = stdout(), ...)
```

Arguments

`x`	an R object.
`con`	a connection for writing the representation to.
`...`	additional arguments to be passed from or to methods.

Details

Graphviz (https://www.graphviz.org) is open source graph visualization software providing several main graph layout programs, of which dot makes "hierarchical" or layered drawings of directed graphs, and hence is typically most suitable for visualizing classification trees.

Using dot, the representation in file 'foo.dot' can be transformed to PostScript or other displayable graphical formats using (a variant of) dot -Tps foo.dot >foo.ps.

Some Weka classifiers (e.g., tree learners such as J48 and M5P) implement a "Drawable" interface providing DOT representations of the fitted models. For such classifiers, the write_to_dot method writes the representation to the specified connection.

---

evaluate_Weka_classifier

Model Statistics for R/Weka Classifiers

Description

Compute model performance statistics for a fitted Weka classifier.

Usage

```
evaluate_Weka_classifier(object, newdata = NULL, cost = NULL,
                         numFolds = 0, complexity = FALSE,
                         class = FALSE, seed = NULL, ...)
```

Arguments

object	a Weka_classifier object.
newdata	an optional data frame in which to look for variables with which to evaluate. If omitted or NULL, the training instances are used.
cost	a square matrix of (mis)classification costs.
numFolds	the number of folds to use in cross-validation.
complexity	option to include entropy-based statistics.
class	option to include class statistics.
seed	optional seed for cross-validation.
...	further arguments passed to other methods (see details).

Details

The function computes and extracts a non-redundant set of performance statistics that is suitable for model interpretation. By default the statistics are computed on the training data.

Currently argument ... only supports the logical variable normalize which tells Weka to normalize the cost matrix so that the cost of a correct classification is zero.

Note that if the class variable is numeric only a subset of the statistics are available. Arguments complexity and class are then not applicable and therefore ignored.

Value

An object of class `Weka_classifier_evaluation`, a list of the following components:

`string`	character, concatenation of the string representations of the performance statistics.
`details`	vector, base statistics, e.g., the percentage of instances correctly classified, etc.
`detailsComplexity`	vector, entropy-based statistics (if selected).
`detailsClass`	matrix, class statistics, e.g., the true positive rate, etc., for each level of the response variable (if selected).
`confusionMatrix`	table, cross-classification of true and predicted classes.

References

I. H. Witten and E. Frank (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. 2nd Edition, Morgan Kaufmann, San Francisco.

Examples

```
## Use some example data.
w <- read.arff(system.file("arff","weather.nominal.arff",
        package = "RWeka"))

## Identify a decision tree.
m <- J48(play~., data = w)
m

## Use 10 fold cross-validation.
e <- evaluate_Weka_classifier(m,
                              cost = matrix(c(0,2,1,0), ncol = 2),
                              numFolds = 10, complexity = TRUE,
                              seed = 123, class = TRUE)
e
summary(e)
e$details
```

`predict_Weka_classifier`

Model Predictions for R/Weka Classifiers

Description

Predicted values based on fitted Weka classifier models.

Usage

```
## S3 method for class 'Weka_classifier'
predict(object, newdata = NULL,
        type = c("class", "probability"), ...)
```

Arguments

object	an object of class inheriting from Weka_classifier.
newdata	an optional data frame in which to look for variables with which to predict. If omitted or NULL, the training instances are used.
type	character string determining whether classes should be predicted (numeric for regression, factor for classification) or class probabilities (only available for classification). May be abbreviated.
...	further arguments passed to or from other methods.

Value

Either a vector with classes or a matrix with the posterior class probabilities, with rows corresponding to instances and columns to classes.

---

predict_Weka_clusterer

Class Predictions for R/Weka Clusterers

Description

Predict class ids or memberships based on fitted Weka clusterers.

Usage

```
## S3 method for class 'Weka_clusterer'
predict(object, newdata = NULL,
        type = c("class_ids", "memberships"), ...)
```

Arguments

object	an object of class inheriting from Weka_clusterer.
newdata	an optional data set for predictions are sought. This must be given for predicting class memberships. If omitted or NULL, the training instances are used for predicting class ids.
type	a character string indicating whether class ids or memberships should be returned. May be abbreviated.
...	further arguments passed to or from other methods.

Details

It is only possible to predict class memberships if the Weka clusterer provides a `distributionForInstance` method.

`read.arff` *Read Data from ARFF Files*

Description

Reads data from Weka Attribute-Relation File Format (ARFF) files.

Usage

```
read.arff(file)
```

Arguments

`file`	a character string with the name of the ARFF file to read from, or a `connection` which will be opened if necessary, and if so closed at the end of the function call.

Value

A data frame containing the data from the ARFF file.

References

Attribute-Relation File Format https://waikato.github.io/weka-wiki/formats_and_processing/arff/

See Also

`write.arff`

Examples

```
read.arff(system.file("arff", "contact-lenses.arff",
                      package = "RWeka"))
```

Weka_associators *R/Weka Associators*

Description

R interfaces to Weka association rule learning algorithms.

Usage

```
Apriori(x, control = NULL)
Tertius(x, control = NULL)
```

Arguments

x	an R object with the data to be associated.
control	an object of class Weka_control, or a character vector of control options, or NULL (default). Available options can be obtained on-line using the Weka Option Wizard WOW, or the Weka documentation.

Details

Apriori implements an Apriori-type algorithm, which iteratively reduces the minimum support until it finds the required number of rules with the given minimum confidence.

Tertius implements a Tertius-type algorithm.

See the references for more information on these algorithms.

Value

A list inheriting from class Weka_associators with components including

associator	a reference (of class jobjRef) to a Java object obtained by applying the Weka buildAssociations method to the training instances using the given control options.

Note

Tertius requires Weka package **tertius** to be installed.

References

R. Agrawal and R. Srikant (1994). Fast algorithms for mining association rules in large databases. *Proceedings of the International Conference on Very Large Databases*, 478–499. Santiago, Chile: Morgan Kaufmann, Los Altos, CA.

P. A. Flach and N. Lachiche (1999). Confirmation-guided discovery of first-order rules with Tertius. *Machine Learning*, **42**, 61–95. doi:10.1023/A:1007656703224.

I. H. Witten and E. Frank (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. 2nd Edition, Morgan Kaufmann, San Francisco.

Examples

```
x <- read.arff(system.file("arff", "contact-lenses.arff",
                           package = "RWeka"))
## Apriori with defaults.
Apriori(x)
## Some options: set required number of rules to 20.
Apriori(x, Weka_control(N = 20))

## Not run:
## Requires Weka package 'tertius' to be installed.
## Tertius with defaults.
Tertius(x)
## Some options: only classification rules (single item in the RHS).
Tertius(x, Weka_control(S = TRUE))

## End(Not run)
```

Weka_attribute_evaluators

R/Weka Attribute Evaluators

Description

R interfaces to Weka attribute evaluators.

Usage

```
GainRatioAttributeEval(formula, data, subset, na.action, control = NULL)
InfoGainAttributeEval(formula, data, subset, na.action, control = NULL)
```

Arguments

formula	a symbolic description of a model. Note that for unsupervised filters the response can be omitted.
data	an optional data frame containing the variables in the model.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
na.action	a function which indicates what should happen when the data contain NAs. See model.frame for details.
control	an object of class Weka_control, or a character vector of control options, or NULL (default). Available options can be obtained on-line using the Weka Option Wizard WOW, or the Weka documentation.

Details

GainRatioAttributeEval evaluates the worth of an attribute by measuring the gain ratio with respect to the class.

InfoGainAttributeEval evaluates the worth of an attribute by measuring the information gain with respect to the class.

Currently, only interfaces to classes which evaluate single attributes (as opposed to subsets, technically, which implement the Weka AttributeEvaluator interface) are possible.

Value

A numeric vector with the figures of merit for the attributes specified by the right hand side of formula.

Examples

```
InfoGainAttributeEval(Species ~ . , data = iris)
```

Weka_classifiers *R/Weka Classifiers*

Description

R interfaces to Weka classifiers.

Details

Supervised learners, i.e., algorithms for classification and regression, are termed "classifiers" by Weka. (Numeric prediction, i.e., regression, is interpreted as prediction of a continuous class.)

R interface functions to Weka classifiers are created by make_Weka_classifier, and have formals formula, data, subset, na.action, and control (default: none), where the first four have the "usual" meanings for statistical modeling functions in R, and the last again specifies the control options to be employed by the Weka learner.

By default, the model formulae should only use the '+' and '-' operators to indicate the variables to be included or not used, respectively.

See model.frame for details on how na.action is used.

Objects created by these interfaces always inherit from class Weka_classifier, and have at least suitable print, summary (via evaluate_Weka_classifier), and predict methods.

See Also

Available "standard" interface functions are documented in Weka_classifier_functions (regression and classification function learners), Weka_classifier_lazy (lazy learners), Weka_classifier_meta (meta learners), Weka_classifier_rules (rule learners), and Weka_classifier_trees (regression and classification tree learners).

Weka_classifier_functions
R/Weka Classifier Functions

Description

R interfaces to Weka regression and classification function learners.

Usage

```
LinearRegression(formula, data, subset, na.action,
                 control = Weka_control(), options = NULL)
Logistic(formula, data, subset, na.action,
         control = Weka_control(), options = NULL)
SMO(formula, data, subset, na.action,
    control = Weka_control(), options = NULL)
```

Arguments

formula	a symbolic description of the model to be fit.
data	an optional data frame containing the variables in the model.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
na.action	a function which indicates what should happen when the data contain NAs. See model.frame for details.
control	an object of class Weka_control giving options to be passed to the Weka learner. Available options can be obtained on-line using the Weka Option Wizard WOW, or the Weka documentation.
options	a named list of further options, or NULL (default). See **Details**.

Details

There are a predict method for predicting from the fitted models, and a summary method based on evaluate_Weka_classifier.

LinearRegression builds suitable linear regression models, using the Akaike criterion for model selection.

Logistic builds multinomial logistic regression models based on ridge estimation (le Cessie and van Houwelingen, 1992).

SMO implements John C. Platt's sequential minimal optimization algorithm for training a support vector classifier using polynomial or RBF kernels. Multi-class problems are solved using pairwise classification.

The model formulae should only use the '+' and '-' operators to indicate the variables to be included or not used, respectively.

Argument options allows further customization. Currently, options model and instances (or partial matches for these) are used: if set to TRUE, the model frame or the corresponding Weka instances, respectively, are included in the fitted model object, possibly speeding up subsequent computations on the object. By default, neither is included.

Value

A list inheriting from classes Weka_functions and Weka_classifiers with components including

classifier	a reference (of class jobjRef) to a Java object obtained by applying the Weka buildClassifier method to build the specified model using the given control options.
predictions	a numeric vector or factor with the model predictions for the training instances (the results of calling the Weka classifyInstance method for the built classifier and each instance).
call	the matched call.

References

J. C. Platt (1998). Fast training of Support Vector Machines using Sequential Minimal Optimization. In B. Schoelkopf, C. Burges, and A. Smola (eds.), *Advances in Kernel Methods — Support Vector Learning*. MIT Press.

I. H. Witten and E. Frank (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. 2nd Edition, Morgan Kaufmann, San Francisco.

See Also

Weka_classifiers

Examples

```
## Linear regression:
## Using standard data set 'mtcars'.
LinearRegression(mpg ~ ., data = mtcars)
## Compare to R:
step(lm(mpg ~ ., data = mtcars), trace = 0)

## Using standard data set 'chickwts'.
LinearRegression(weight ~ feed, data = chickwts)
## (Note the interactions!)

## Logistic regression:
## Using standard data set 'infert'.
STATUS <- factor(infert$case, labels = c("control", "case"))
Logistic(STATUS ~ spontaneous + induced, data = infert)
## Compare to R:
glm(STATUS ~ spontaneous + induced, data = infert, family = binomial())

## Sequential minimal optimization algorithm for training a support
## vector classifier, using am RBF kernel with a non-default gamma
```

```
## parameter (argument '-G') instead of the default polynomial kernel
## (from a question on r-help):
SMO(Species ~ ., data = iris,
    control = Weka_control(K =
    list("weka.classifiers.functions.supportVector.RBFKernel", G = 2)))
## In fact, by some hidden magic it also "works" to give the "base" name
## of the Weka kernel class:
SMO(Species ~ ., data = iris,
    control = Weka_control(K = list("RBFKernel", G = 2)))
```

`Weka_classifier_lazy` *R/Weka Lazy Learners*

Description

R interfaces to Weka lazy learners.

Usage

```
IBk(formula, data, subset, na.action,
    control = Weka_control(), options = NULL)
LBR(formula, data, subset, na.action,
    control = Weka_control(), options = NULL)
```

Arguments

`formula`	a symbolic description of the model to be fit.
`data`	an optional data frame containing the variables in the model.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`na.action`	a function which indicates what should happen when the data contain `NA`s. See `model.frame` for details.
`control`	an object of class `Weka_control` giving options to be passed to the Weka learner. Available options can be obtained on-line using the Weka Option Wizard `WOW`, or the Weka documentation.
`options`	a named list of further options, or `NULL` (default). See **Details**.

Details

There are a `predict` method for predicting from the fitted models, and a `summary` method based on `evaluate_Weka_classifier`.

`IBk` provides a k-nearest neighbors classifier, see Aha & Kibler (1991).

`LBR` ("Lazy Bayesian Rules") implements a lazy learning approach to lessening the attribute-independence assumption of naive Bayes as suggested by Zheng & Webb (2000).

The model formulae should only use the '`+`' and '`-`' operators to indicate the variables to be included or not used, respectively.

Argument options allows further customization. Currently, options model and instances (or partial matches for these) are used: if set to TRUE, the model frame or the corresponding Weka instances, respectively, are included in the fitted model object, possibly speeding up subsequent computations on the object. By default, neither is included.

Value

A list inheriting from classes Weka_lazy and Weka_classifiers with components including

classifier	a reference (of class jobjRef) to a Java object obtained by applying the Weka buildClassifier method to build the specified model using the given control options.
predictions	a numeric vector or factor with the model predictions for the training instances (the results of calling the Weka classifyInstance method for the built classifier and each instance).
call	the matched call.

Note

LBR requires Weka package **lazyBayesianRules** to be installed.

References

D. Aha and D. Kibler (1991). Instance-based learning algorithms. *Machine Learning*, **6**, 37–66. doi:10.1007/BF00153759.

Z. Zheng and G. Webb (2000). Lazy learning of Bayesian rules. *Machine Learning*, **41**/1, 53–84. doi:10.1023/A:1007613203719.

See Also

Weka_classifiers

Weka_classifier_meta *R/Weka Meta Learners*

Description

R interfaces to Weka meta learners.

Usage

```
AdaBoostM1(formula, data, subset, na.action,
           control = Weka_control(), options = NULL)
Bagging(formula, data, subset, na.action,
        control = Weka_control(), options = NULL)
LogitBoost(formula, data, subset, na.action,
           control = Weka_control(), options = NULL)
```

```
MultiBoostAB(formula, data, subset, na.action,
             control = Weka_control(), options = NULL)
Stacking(formula, data, subset, na.action,
         control = Weka_control(), options = NULL)
CostSensitiveClassifier(formula, data, subset, na.action,
                        control = Weka_control(), options = NULL)
```

Arguments

formula	a symbolic description of the model to be fit.
data	an optional data frame containing the variables in the model.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
na.action	a function which indicates what should happen when the data contain NAs. See model.frame for details.
control	an object of class Weka_control giving options to be passed to the Weka learner. Available options can be obtained on-line using the Weka Option Wizard WOW, or the Weka documentation. Base classifiers with an available R/Weka interface (see list_Weka_interfaces), can be specified (using the 'W' option) via their "base name" as shown in the interface registry (see the examples), or their interface function.
options	a named list of further options, or NULL (default). See **Details**.

Details

There are a predict method for predicting from the fitted models, and a summary method based on evaluate_Weka_classifier.

AdaBoostM1 implements the AdaBoost M1 method of Freund and Schapire (1996).

Bagging provides bagging (Breiman, 1996).

LogitBoost performs boosting via additive logistic regression (Friedman, Hastie and Tibshirani, 2000).

MultiBoostAB implements MultiBoosting (Webb, 2000), an extension to the AdaBoost technique for forming decision committees which can be viewed as a combination of AdaBoost and "wagging".

Stacking provides stacking (Wolpert, 1992).

CostSensitiveClassifier makes its base classifier cost-sensitive.

The model formulae should only use the '+' and '-' operators to indicate the variables to be included or not used, respectively.

Argument options allows further customization. Currently, options model and instances (or partial matches for these) are used: if set to TRUE, the model frame or the corresponding Weka instances, respectively, are included in the fitted model object, possibly speeding up subsequent computations on the object. By default, neither is included.

Value

A list inheriting from classes `Weka_meta` and `Weka_classifiers` with components including

`classifier`	a reference (of class `jobjRef`) to a Java object obtained by applying the Weka `buildClassifier` method to build the specified model using the given control options.
`predictions`	a numeric vector or factor with the model predictions for the training instances (the results of calling the Weka `classifyInstance` method for the built classifier and each instance).
`call`	the matched call.

Note

`multiBoostAB` requires Weka package **multiBoostAB** to be installed.

References

L. Breiman (1996). Bagging predictors. *Machine Learning*, **24**/2, 123–140. doi:10.1023/A:1018054314350.

Y. Freund and R. E. Schapire (1996). Experiments with a new boosting algorithm. In *Proceedings of the International Conference on Machine Learning*, pages 148–156. Morgan Kaufmann: San Francisco.

J. H. Friedman, T. Hastie, and R. Tibshirani (2000). Additive logistic regression: A statistical view of boosting. *Annals of Statistics*, **28**/2, 337–374. doi:10.1214/aos/1016218223.

G. I. Webb (2000). MultiBoosting: A technique for combining boosting and wagging. *Machine Learning*, **40**/2, 159–196. doi:10.1023/A:1007659514849.

I. H. Witten and E. Frank (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. 2nd Edition, Morgan Kaufmann, San Francisco.

D. H. Wolpert (1992). Stacked generalization. *Neural Networks*, **5**, 241–259. doi:10.1016/S0893-6080(05)800231.

See Also

Weka_classifiers

Examples

```
## Use AdaBoostM1 with decision stumps.
m1 <- AdaBoostM1(Species ~ ., data = iris,
                 control = Weka_control(W = "DecisionStump"))
table(predict(m1), iris$Species)

summary(m1) # uses evaluate_Weka_classifier()

## Control options for the base classifiers employed by the meta
## learners (apart from Stacking) can be given as follows:
m2 <- AdaBoostM1(Species ~ ., data = iris,
                 control = Weka_control(W = list(J48, M = 30)))
```

Weka_classifier_rules *R/Weka Rule Learners*

Description

R interfaces to Weka rule learners.

Usage

```
JRip(formula, data, subset, na.action,
     control = Weka_control(), options = NULL)
M5Rules(formula, data, subset, na.action,
        control = Weka_control(), options = NULL)
OneR(formula, data, subset, na.action,
     control = Weka_control(), options = NULL)
PART(formula, data, subset, na.action,
     control = Weka_control(), options = NULL)
```

Arguments

formula	a symbolic description of the model to be fit.
data	an optional data frame containing the variables in the model.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
na.action	a function which indicates what should happen when the data contain NAs. See model.frame for details.
control	an object of class Weka_control giving options to be passed to the Weka learner. Available options can be obtained on-line using the Weka Option Wizard WOW, or the Weka documentation.
options	a named list of further options, or NULL (default). See **Details**.

Details

There are a predict method for predicting from the fitted models, and a summary method based on evaluate_Weka_classifier.

JRip implements a propositional rule learner, "Repeated Incremental Pruning to Produce Error Reduction" (RIPPER), as proposed by Cohen (1995).

M5Rules generates a decision list for regression problems using separate-and-conquer. In each iteration it builds an model tree using M5 and makes the "best" leaf into a rule. See Hall, Holmes and Frank (1999) for more information.

OneR builds a simple 1-R classifier, see Holte (1993).

PART generates PART decision lists using the approach of Frank and Witten (1998).

The model formulae should only use the '+' and '-' operators to indicate the variables to be included or not used, respectively.

Argument options allows further customization. Currently, options model and instances (or partial matches for these) are used: if set to TRUE, the model frame or the corresponding Weka instances, respectively, are included in the fitted model object, possibly speeding up subsequent computations on the object. By default, neither is included.

Value

A list inheriting from classes Weka_rules and Weka_classifiers with components including

classifier	a reference (of class jobjRef) to a Java object obtained by applying the Weka buildClassifier method to build the specified model using the given control options.
predictions	a numeric vector or factor with the model predictions for the training instances (the results of calling the Weka classifyInstance method for the built classifier and each instance).
call	the matched call.

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. doi:10.1016/B9781558603776.500232.

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. https://www.cs.waikato.ac.nz/~eibe/pubs/ML98-57.ps.gz

M. Hall, G. Holmes, and E. Frank (1999). Generating rule sets from model trees. *Proceedings of the Twelfth Australian Joint Conference on Artificial Intelligence*, Sydney, Australia, pages 1–12. Springer-Verlag. https://www.cs.waikato.ac.nz/~eibe/pubs/ajc.pdf

R. C. Holte (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, **11**, 63–91. doi:10.1023/A:1022631118932.

I. H. Witten and E. Frank (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. 2nd Edition, Morgan Kaufmann, San Francisco.

See Also

Weka_classifiers

Examples

```
M5Rules(mpg ~ ., data = mtcars)

m <- PART(Species ~ ., data = iris)
m
summary(m)
```

`Weka_classifier_trees` *R/Weka Classifier Trees*

Description

R interfaces to Weka regression and classification tree learners.

Usage

```
J48(formula, data, subset, na.action,
    control = Weka_control(), options = NULL)
LMT(formula, data, subset, na.action,
    control = Weka_control(), options = NULL)
M5P(formula, data, subset, na.action,
    control = Weka_control(), options = NULL)
DecisionStump(formula, data, subset, na.action,
              control = Weka_control(), options = NULL)
```

Arguments

`formula`	a symbolic description of the model to be fit.
`data`	an optional data frame containing the variables in the model.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`na.action`	a function which indicates what should happen when the data contain `NA`s. See `model.frame` for details.
`control`	an object of class `Weka_control` giving options to be passed to the Weka learner. Available options can be obtained on-line using the Weka Option Wizard `WOW`, or the Weka documentation.
`options`	a named list of further options, or `NULL` (default). See **Details**.

Details

There are a `predict` method for predicting from the fitted models, and a `summary` method based on `evaluate_Weka_classifier`.

There is also a `plot` method for fitted binary `Weka_trees` via the facilities provided by package **partykit**. This converts the `Weka_tree` to a `party` object and then simply calls the plot method of this class (see `plot.party`).

Provided the Weka classification tree learner implements the "Drawable" interface (i.e., provides a `graph` method), `write_to_dot` can be used to create a DOT representation of the tree for visualization via Graphviz or the **Rgraphviz** package.

`J48` generates unpruned or pruned C4.5 decision trees (Quinlan, 1993).

`LMT` implements "Logistic Model Trees" (Landwehr, 2003; Landwehr et al., 2005).

M5P (where the 'P' stands for 'prime') generates M5 model trees using the M5' algorithm, which was introduced in Wang & Witten (1997) and enhances the original M5 algorithm by Quinlan (1992).

DecisionStump implements decision stumps (trees with a single split only), which are frequently used as base learners for meta learners such as Boosting.

The model formulae should only use the '+' and '-' operators to indicate the variables to be included or not used, respectively.

Argument options allows further customization. Currently, options model and instances (or partial matches for these) are used: if set to TRUE, the model frame or the corresponding Weka instances, respectively, are included in the fitted model object, possibly speeding up subsequent computations on the object. By default, neither is included.

parse_Weka_digraph can parse the graph associated with a Weka tree classifier (and obtained by invoking its graph() method in Weka), returning a simple list with nodes and edges.

Value

A list inheriting from classes Weka_tree and Weka_classifiers with components including

classifier	a reference (of class jobjRef) to a Java object obtained by applying the Weka buildClassifier method to build the specified model using the given control options.
predictions	a numeric vector or factor with the model predictions for the training instances (the results of calling the Weka classifyInstance method for the built classifier and each instance).
call	the matched call.

References

N. Landwehr (2003). *Logistic Model Trees*. Master's thesis, Institute for Computer Science, University of Freiburg, Germany. https://www.cs.uni-potsdam.de/ml/landwehr/diploma_thesis.pdf

N. Landwehr, M. Hall, and E. Frank (2005). Logistic Model Trees. *Machine Learning*, **59**, 161–205. doi:10.1007/s1099400504663.

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

R. Quinlan (1992). Learning with continuous classes. *Proceedings of the Australian Joint Conference on Artificial Intelligence*, 343–348. World Scientific, Singapore.

Y. Wang and I. H. Witten (1997). Induction of model trees for predicting continuous classes. *Proceedings of the European Conference on Machine Learning*. University of Economics, Faculty of Informatics and Statistics, Prague.

I. H. Witten and E. Frank (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. 2nd Edition, Morgan Kaufmann, San Francisco.

See Also

Weka_classifiers

Examples

```
m1 <- J48(Species ~ ., data = iris)

## print and summary
m1
summary(m1) # calls evaluate_Weka_classifier()
table(iris$Species, predict(m1)) # by hand

## visualization
## use partykit package
if(require("partykit", quietly = TRUE)) plot(m1)
## or Graphviz
write_to_dot(m1)
## or Rgraphviz
## Not run:
library("Rgraphviz")
ff <- tempfile()
write_to_dot(m1, ff)
plot(agread(ff))

## End(Not run)

## Using some Weka data sets ...

## J48
DF2 <- read.arff(system.file("arff", "contact-lenses.arff",
                             package = "RWeka"))
m2 <- J48(`contact-lenses` ~ ., data = DF2)
m2
table(DF2$`contact-lenses`, predict(m2))
if(require("partykit", quietly = TRUE)) plot(m2)

## M5P
DF3 <- read.arff(system.file("arff", "cpu.arff", package = "RWeka"))
m3 <- M5P(class ~ ., data = DF3)
m3
if(require("partykit", quietly = TRUE)) plot(m3)

## Logistic Model Tree.
DF4 <- read.arff(system.file("arff", "weather.arff", package = "RWeka"))
m4 <- LMT(play ~ ., data = DF4)
m4
table(DF4$play, predict(m4))

## Larger scale example.
if(require("mlbench", quietly = TRUE)
   && require("partykit", quietly = TRUE)) {
    ## Predict diabetes status for Pima Indian women
    data("PimaIndiansDiabetes", package = "mlbench")
    ## Fit J48 tree with reduced error pruning
    m5 <- J48(diabetes ~ ., data = PimaIndiansDiabetes,
              control = Weka_control(R = TRUE))
```

```
    plot(m5)
    ## (Make sure that the plotting device is big enough for the tree.)
}
```

`Weka_clusterers` *R/Weka Clusterers*

Description

R interfaces to Weka clustering algorithms.

Usage

```
Cobweb(x, control = NULL)
FarthestFirst(x, control = NULL)
SimpleKMeans(x, control = NULL)
XMeans(x, control = NULL)
DBScan(x, control = NULL)
```

Arguments

`x`	an R object with the data to be clustered.
`control`	an object of class `Weka_control`, or a character vector of control options, or `NULL` (default). Available options can be obtained on-line using the Weka Option Wizard `WOW`, or the Weka documentation.

Details

There is a `predict` method for predicting class ids or memberships from the fitted clusterers.

`Cobweb` implements the Cobweb (Fisher, 1987) and Classit (Gennari et al., 1989) clustering algorithms.

`FarthestFirst` provides the "farthest first traversal algorithm" by Hochbaum and Shmoys, which works as a fast simple approximate clusterer modeled after simple k-means.

`SimpleKMeans` provides clustering with the k-means algorithm.

`XMeans` provides k-means extended by an "Improve-Structure part" and automatically determines the number of clusters.

`DBScan` provides the "density-based clustering algorithm" by Ester, Kriegel, Sander, and Xu. Note that noise points are assigned to `NA`.

Value

A list inheriting from class `Weka_clusterers` with components including

`clusterer`	a reference (of class `jobjRef`) to a Java object obtained by applying the Weka `buildClusterer` method to the training instances using the given control options.

`class_ids` a vector of integers indicating the class to which each training instance is allocated (the results of calling the Weka `clusterInstance` method for the built clusterer and each instance).

Note

`XMeans` requires Weka package **XMeans** to be installed.

`DBScan` requires Weka package **optics_dbScan** to be installed.

References

M. Ester, H.-P. Kriegel, J. Sander, and X. Xu (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*, Portland, OR, 226–231. AAAI Press.

D. H. Fisher (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, **2**/2, 139–172. doi:10.1023/A:1022852608280.

J. Gennari, P. Langley, and D. H. Fisher (1989). Models of incremental concept formation. *Artificial Intelligence*, **40**, 11–62.

D. S. Hochbaum and D. B. Shmoys (1985). A best possible heuristic for the k-center problem, *Mathematics of Operations Research*, **10**(2), 180–184. doi:10.1287/moor.10.2.180.

D. Pelleg and A. W. Moore (2006). X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In: *Seventeenth International Conference on Machine Learning*, 727–734. Morgan Kaufmann.

I. H. Witten and E. Frank (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. 2nd Edition, Morgan Kaufmann, San Francisco.

Examples

```
cl1 <- SimpleKMeans(iris[, -5], Weka_control(N = 3))
cl1
table(predict(cl1), iris$Species)

## Not run:
## Requires Weka package 'XMeans' to be installed.
## Use XMeans with a KDTree.
cl2 <- XMeans(iris[, -5],
              c("-L", 3, "-H", 7, "-use-kdtree",
                "-K", "weka.core.neighboursearch.KDTree -P"))
cl2
table(predict(cl2), iris$Species)

## End(Not run)
```

Weka_control *Control Weka Options*

Description

Set control options for Weka learners.

Usage

```
Weka_control(...)
```

Arguments

...	named arguments of control options, see the details and examples.

Details

The available options for a Weka learner, `foo()` say, can be queried by `WOW(foo)` and then conveniently set by `Weka_control()`. See below for an example.

One can use lists for options taking multiple arguments, see the documentation for `SMO` for an example.

Value

A list of class `Weka_control` which can be coerced to `character` for passing it to Weka.

See Also

`WOW`

Examples

```
## Query J4.8 options:
WOW("J48")
## Learn J4.8 tree on iris data with default settings:
J48(Species ~ ., data = iris)
## Learn J4.8 tree with reduced error pruning (-R) and
## minimum number of instances set to 5 (-M 5):
J48(Species ~ ., data = iris, control = Weka_control(R = TRUE, M = 5))
```

Weka_converters *R/Weka File Loaders and Savers*

Description

R interfaces to Weka file loaders and savers.

Usage

```
C45Loader(file)
XRFFLoader(file)
C45Saver(x, file, control = NULL)
XRFFSaver(x, file, control = NULL)
```

Arguments

file	a non-empty character string naming a file to read from or write to.
x	the data to be written, preferably a matrix or data frame. If not, coercion to a data frame is attempted.
control	an object of class Weka_control, or a character vector of control options, or NULL (default). Available options can be obtained on-line using the Weka Option Wizard WOW, or the Weka documentation.

Details

C45Loader and C45Saver use the format employed by the C4.5 algorithm/software, where data is stored in two separate '.names' and '.data' files.

XRFFLoader and XRFFSaver handle XRFF (eXtensible attribute-Relation File Format, an XML-based extension of Weka's native Attribute-Relation File Format) files.

Value

Invisibly NULL for the savers.

A data frame containing the data from the given file for the loaders.

See Also

read.arff, write.arff.

Weka_filters *R/Weka Filters*

Description

R interfaces to Weka filters.

Usage

```
Normalize(formula, data, subset, na.action, control = NULL)
Discretize(formula, data, subset, na.action, control = NULL)
```

Arguments

formula	a symbolic description of a model. Note that for unsupervised filters the response can be omitted.
data	an optional data frame containing the variables in the model.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
na.action	a function which indicates what should happen when the data contain NAs. See model.frame for details.
control	an object of class Weka_control, or a character vector of control options, or NULL (default). Available options can be obtained on-line using the Weka Option Wizard WOW, or the Weka documentation.

Details

Normalize implements an unsupervised filter that normalizes all instances of a dataset to have a given norm. Only numeric values are considered, and the class attribute is ignored.

Discretize implements a supervised instance filter that discretizes a range of numeric attributes in the dataset into nominal attributes. Discretization is by Fayyad & Irani's MDL method (the default).

Note that these methods ignore nominal attributes, i.e., variables of class factor.

Value

A data frame.

References

U. M. Fayyad and K. B. Irani (1993). Multi-interval discretization of continuous-valued attributes for classification learning. *Thirteenth International Joint Conference on Artificial Intelligence*, 1022–1027. Morgan Kaufmann.

I. H. Witten and E. Frank (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. 2nd Edition, Morgan Kaufmann, San Francisco.

Examples

```
## Using a Weka data set ...
w <- read.arff(system.file("arff","weather.arff",
        package = "RWeka"))

## Normalize (response irrelevant)
m1 <- Normalize(~., data = w)
m1

## Discretize
m2 <- Discretize(play ~., data = w)
m2
```

`Weka_interfaces` *R/Weka interfaces*

Description

Create an R interface to an existing Weka learner, attribute evaluator or filter, or show the available interfaces.

Usage

```
make_Weka_associator(name, class = NULL,
                     init = NULL, package = NULL)
make_Weka_attribute_evaluator(name, class = NULL,
                              init = NULL, package = NULL)
make_Weka_classifier(name, class = NULL, handlers = list(),
                     init = NULL, package = NULL)
make_Weka_clusterer(name, class = NULL,
                    init = NULL, package = NULL)
make_Weka_filter(name, class = NULL,
                 init = NULL, package = NULL)
list_Weka_interfaces()
make_Weka_package_loader(p)
```

Arguments

`name`	a character string giving the fully qualified name of a Weka learner/filter class in JNI notation.
`class`	`NULL` (default), or a character vector giving the names of R classes the objects returned by the interface function should inherit from in addition to the default ones (for representing associators, classifiers, and clusterers).
`handlers`	a named list of special handler functions, see **Details**.
`init`	`NULL`, or a function with no arguments to be called when the interface is used for building the learner/filter, or queried for available options via `WOW`. Typically, this is used for loading Weka packages when interfacing functionality in these.

package	NULL (default), or a character string giving the name of the external Weka package providing the learner/filter class specified by name.
p	a character string naming a Weka package to be loaded via WPM.

Details

make_Weka_associator and make_Weka_clusterer create an R function providing an interface to a Weka association learner or a Weka clusterer, respectively. This interface function has formals x and control = NULL, representing the training instances and control options to be employed. Objects created by these interface functions always inherit from classes Weka_associator and Weka_clusterer, respectively, and have at least suitable print methods. Fitted clusterers also have a predict method.

make_Weka_classifier creates an interface function for a Weka classifier, with formals formula, data, subset, na.action, and control (default: none), where the first four have the "usual" meanings for statistical modeling functions in R, and the last again specifies the control options to be employed by the Weka learner. Objects created by these interfaces always inherit from class Weka_classifier, and have at least suitable print and predict methods.

make_Weka_filter creates an interface function for a Weka filter, with formals formula, data, subset, na.action, and control = NULL, where the first four have the "usual" meanings for statistical modeling functions in R, and the last again specifies the control options to be employed by the Weka filter. Note that the response variable can be omitted from formula if the filter is "unsupervised". Objects created by these interface functions are (currently) always of class data.frame.

make_Weka_attribute_evaluator creates an interface function for a Weka attribute evaluation class which implements the AttributeEvaluator interface, with formals as for the classifier interface functions.

Certain aspects of the interface function can be customized by providing handlers. Currently, only *control* handlers (functions given as the control component of the list of handlers) are used for processing the given control arguments before passing them to the Weka classifier. This is used, e.g., by the meta learners to allow the specification of registered base learners by their "base names" (rather their full Weka/Java class names).

In addition to creating interface functions, the interfaces are registered (under the name of the Weka class interfaced), which in particular allows the Weka Option Wizard (WOW) to conveniently give on-line information about available control options for the interfaces.

list_Weka_interfaces lists the *available* interfaces.

Finally, make_Weka_package_loader generates init hooks for loading required and already installed Weka packages.

It is straightforward to register new interfaces in addition to the ones package **RWeka** provides by default.

References

K. Hornik, C. Buchta, and A. Zeileis (2009). Open-source machine learning: R meets Weka. *Computational Statistics*, **24**/2, 225–232. doi:10.1007/s0018000801197.

Examples

```
## Create an interface to Weka's Naive Bayes classifier.
NB <- make_Weka_classifier("weka/classifiers/bayes/NaiveBayes")
## Note that this has a very useful print method:
NB
## And we can use the Weka Option Wizard for finding out more:
WOW(NB)
## And actually use the interface ...
if(require("e1071", quietly = TRUE) &&
   require("mlbench", quietly = TRUE)) {
    data("HouseVotes84", package = "mlbench")
    model <- NB(Class ~ ., data = HouseVotes84)
    predict(model, HouseVotes84[1:10, -1])
    predict(model, HouseVotes84[1:10, -1], type = "prob")
}
## (Compare this to David Meyer's naiveBayes() in package 'e1071'.)
```

`Weka_stemmers` *R/Weka Stemmers*

Description

R interfaces to Weka stemmers.

Usage

```
IteratedLovinsStemmer(x, control = NULL)
LovinsStemmer(x, control = NULL)
```

Arguments

`x`	a character vector with words to be stemmed.
`control`	an object of class `Weka_control`, or a character vector of control options, or `NULL` (default). Available options can be obtained on-line using the Weka Option Wizard `WOW`, or the Weka documentation.

Value

A character vector with the stemmed words.

References

J. B. Lovins (1968), Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, **11**, 22–31.

Weka_tokenizers *R/Weka Tokenizers*

Description

R interfaces to Weka tokenizers.

Usage

```
AlphabeticTokenizer(x, control = NULL)
NGramTokenizer(x, control = NULL)
WordTokenizer(x, control = NULL)
```

Arguments

x	a character vector with strings to be tokenized.
control	an object of class Weka_control, or a character vector of control options, or NULL (default). Available options can be obtained on-line using the Weka Option Wizard WOW, or the Weka documentation.

Details

AlphabeticTokenizer is an alphabetic string tokenizer, where tokens are to be formed only from contiguous alphabetic sequences.

NGramTokenizer splits strings into n-grams with given minimal and maximal numbers of grams.

WordTokenizer is a simple word tokenizer.

Value

A character vector with the tokenized strings.

WOW *Weka Option Wizard*

Description

Give on-line information about available control options for Weka learners or filters and their R interfaces.

Usage

```
WOW(x)
```

Arguments

x	a character string giving either the fully qualified name of a Weka learner or filter class in JNI notation, or the name of an available R interface, or an object obtained from applying these interfaces to build an associator, classifier, clusterer, or filter.

Details

See `list_Weka_interfaces` for the available interface functions.

References

K. Hornik, C. Buchta, and A. Zeileis (2009). Open-source machine learning: R meets Weka. *Computational Statistics*, **24**/2, 225–232. doi:10.1007/s0018000801197.

Examples

```
## The name of an "existing" (registered) interface.
WOW("J48")
## The name of some Weka class (not necessarily in the interface
## registry):
WOW("weka/classifiers/bayes/NaiveBayes")
```

WPM *Weka Package Manager*

Description

Manage Weka packages.

Usage

```
WPM(cmd, ...)
```

Arguments

cmd	a character string specifying the action to be performed. Must be one of `"refresh-cache"`, `"list-packages"`, `"package-info"`, `"install-package"`, `"remove-package"`, `"toggle-load-status"` or `"load-packages"` (or a unique abbreviation thereof).
...	character strings giving further arguments required for the action to be performed. See **Details**.

Details

Available actions and respective additional arguments are as follows.

`"refresh-cache"` Refresh the cached copy of the package meta data from the central package repository.

`"list-packages"` print information (version numbers and short descriptions) about packages as specified by an additional keyword which must be one of `"all"` (all packages the system knows about), `"installed"` (all packages installed locally), or (`"available"` (all known packages not installed locally), or a unique abbreviation thereof.

`"package-info"` print information (metadata) about a package. Requires two additional character string arguments: a keyword and the package name. The keyword must be one of `"repository"` (print info from the repository) or `"installed"` (print info on the installed version), or a unique abbreviation thereof.

`"install-package"` install a package as specified by an additional character string giving its name. (In principle, one could also provide a file path or URL to a zip file.)

`"remove-package"` remove a given (installed) package.

`"toggle-load-status"` toggle the load status of the given (installed) packages.

`"load-packages"` load all installed packages with active load status.

Note

Weka stores packages and their information in the Weka home directory, as given by the value of the environment variable `WEKA_HOME`; if this is not set, the '`wekafiles`' subdirectory of the user's home directory is used. If this Weka home directory was not created yet, `WPM()` will instead use a temporary directory in the R session directory: to achieve persistence, users need to create the Weka home directory before using `WPM()`.

Examples

```
## Not run:
## Start by building/refreshing the cache.
WPM("refresh-cache")
## Show the packages installed locally.
WPM("list-packages", "installed")
## Show the packages available from the central Weka package
## repository and not installed locally.
WPM("list-packages", "available")
## Show repository information about package XMeans.
WPM("package-info", "repository", "XMeans")

## End(Not run)
```

`write.arff` *Write Data into ARFF Files*

Description

Writes data into Weka Attribute-Relation File Format (ARFF) files.

Usage

```
write.arff(x, file, eol = "\n")
```

Arguments

`x`	the data to be written, preferably a matrix or data frame. If not, coercion to a data frame is attempted.
`file`	either a character string naming a file, or a connection. `""` indicates output to the standard output connection.
`eol`	the character(s) to print at the end of each line (row).

References

Attribute-Relation File Format https://waikato.github.io/weka-wiki/formats_and_processing/arff/

See Also

`read.arff`

Examples

```
write.arff(iris, file = "")
```

Index