Package: pmml (via r-universe)

February 26, 2025

Type Package

Title Generate PMML for Various Models

Version 2.5.2

Depends XML

Suggests ada, amap, arules, caret, clue, data.table, forecast, gbm, glmnet, Matrix, neighbr, nnet, rpart, randomForest, rattle, kernlab, e1071, testthat, survival, xgboost, knitr, rmarkdown, covr, tibble

Imports methods, stats, utils, stringr

License GPL-3 | file LICENSE

Description The Predictive Model Markup Language (PMML) is an XML-based language which provides a way for applications to define machine learning, statistical and data mining models and to share models between PMML compliant applications. More information about the PMML industry standard and the Data Mining Group can be found at <http://dmg.org/>. The generated PMML can be imported into any PMML consuming application, such as Zementis Predictive Analytics products. The package isofor (used for anomaly detection) can be installed with devtools::install_github(``gravesee/isofor").

URL https://open-source.softwareag.com/r-pmml/, https://github.com/SoftwareAG/r-pmml, https://www.softwareag.com/corporate/products/az/zementis/default.html

BugReports https://github.com/SoftwareAG/r-pmml/issues

NeedsCompilation no

RoxygenNote 7.1.2

VignetteBuilder knitr

Encoding UTF-8

Config/pak/sysreqs libicu-dev libxml2-dev

Repository https://cumulocity-iot.r-universe.dev

RemoteUrl https://github.com/cumulocity-iot/r-pmml

RemoteRef HEAD

RemoteSha 9daa76510cf9e47084e33539b30068fe45a6c41a

Contents

`add_attributes` *Add attribute values to an existing element in a given PMML file.*

Description

Add attribute values to an existing element in a given PMML file.

Usage

```
add_attributes(
  xml_model = NULL,
  xpath = NULL,
  attributes = NULL,
  namespace = "4_4",
  ...
)
```

Arguments

`xml_model`	The PMML model in a XML node format. If the model is a text file, it should be converted to an XML node, for example, using the file_to_xml_node function.
`xpath`	The XPath to the element to which the attributes are to be added.
`attributes`	The attributes to be added to the data fields. The user should make sure that the attributes being added are allowed in the PMML schema.
`namespace`	The namespace of the PMML model. This is frequently also the PMML version of the model.
`...`	Further arguments passed to or from other methods.

Details

Add attributes to an arbitrary XML element. This is an experimental function designed to be more general than the 'add_mining_field_attributes' and 'add_data_field_attributes' functions.

The attribute information can be provided as a vector. Multiple attribute names and values can be passes as vector elements to enable inserting multiple attributes. However, this function overwrites any pre-existing attribute values, so it must be used with care. This behavior is by design as this feature is meant to help an user add new defined attribute values at different times. The XPath has to include the namespace as shown in the examples.

Value

An object of class `XMLNode` as that defined by the **XML** package. This represents the top level, or root node, of the XML document and is of type PMML. It can be written to file with `saveXML`.

Author(s)

Tridivesh Jena

Examples

```
# Make a sample model:
fit <- lm(Sepal.Length ~ ., data = iris[, -5])
fit_pmml <- pmml(fit)

# Add arbitrary attributes to the 1st 'NumericPredictor' element. The
# attributes are for demostration only (they are not allowed under
# the PMML schema). The command assumes the default namespace.
fit_pmml_2 <- add_attributes(fit_pmml, "/p:PMML/descendant::p:NumericPredictor[1]",
  attributes = c(a = 1, b = "b")
)

# Add attributes to the NumericPredictor element which has
# 'Petal.Length' as the 'name' attribute:
fit_pmml_3 <- add_attributes(fit_pmml,
  "/p:PMML/descendant::p:NumericPredictor[@name='Petal.Length']",
  attributes = c(a = 1, b = "b")
)

# 3 NumericElements exist which have '1' as the 'exponent' attribute.
# Add new attributes to the 3rd one:
fit_pmml_4 <- add_attributes(fit_pmml,
  "/p:PMML/descendant::p:NumericPredictor[@exponent='1'][3]",
  attributes = c(a = 1, b = "b")
)

# Add attributes to the 1st element whose 'name' attribute contains
# 'Length':
fit_pmml_5 <- add_attributes(fit_pmml,
  "/p:PMML/descendant::p:NumericPredictor[contains(@name,'Length')]",
  attributes = c(a = 1, b = "b")
)
```

`add_data_field_attributes` *Add attribute values to an existing DataField element in a given PMML file*

Description

Add attribute values to an existing DataField element in a given PMML file

Usage

```
add_data_field_attributes(
  xml_model = NULL,
  attributes = NULL,
  field = NULL,
  namespace = "4_4",
  ...
)
```

Arguments

xml_model	The PMML model in a XML node format. If the model is a text file, it should be converted to an XML node, for example, using the file_to_xml_node function.
attributes	The attributes to be added to the data fields. The user should make sure that the attributes being added are allowed in the PMML schema.
field	The field to which the attributes are to be added. This is used when the attributes are a vector of name-value pairs, intended for this one field.
namespace	The namespace of the PMML model. This is frequently also the PMML version of the model.
...	Further arguments passed to or from other methods.

Details

The PMML schema allows a DataField element to have various attributes, which, although useful, may not always be present in a PMML model. This function makes it possible to add such attributes to DataFields of an existing PMML file.

The attribute information can be provided as a dataframe or a vector. Each row of the data frame corresponds to an attribute name and each column corresponding to a variable name. This way one can add as many attributes to as many variables as one wants in one step. A more convenient method to add multiple attributes to one field might be to give the attribute name and values as a vector. This function may be used multiple times to add new attribute values step-by-step. However this function overwrites any pre-existing attribute values, so it must be used with care. This behavior is by design as this feature is meant to help an user add new defined attribute values at different times. For example, one may use this to modify the display name of a field at different times.

Value

An object of class XMLNode as that defined by the **XML** package. This represents the top level, or root node, of the XML document and is of type PMML. It can be written to file with saveXML.

Author(s)

Tridivesh Jena

Examples

```
# Make a sample model:
fit <- lm(Sepal.Length ~ ., data = iris[, -5])
fit_pmml <- pmml(fit)

# The resulting model has mining fields with no information besides
# fieldName, dataType and optype. This object is already an xml
# node (not an external text file), so there is no need to convert
# it to an xml node object.

# Create data frame with attribute information:

attributes <- data.frame(c("FlowerWidth", 1), c("FlowerLength", 0),
  stringsAsFactors = FALSE
)
rownames(attributes) <- c("displayName", "isCyclic")
colnames(attributes) <- c("Sepal.Width", "Petal.Length")

# Although not needed in this first try, necessary to easily add
# new values later. Removes values as factors so that new values
# added later are not evaluated as factor values and thus rejected
# as invalid.
attributes[] <- lapply(attributes, as.character)

fit_pmml_2 <- add_data_field_attributes(fit_pmml,
  attributes,
  namespace = "4_4"
)

# Alternative method to add attributes to a single field,
# "Sepal.Width":
fit_pmml_3 <- add_data_field_attributes(
  fit_pmml, c(displayName = "FlowerWidth", isCyclic = 1),
  "Sepal.Width"
)


mi <- make_intervals(
  list("openClosed", "closedClosed", "closedOpen"),
  list(NULL, 1, 2), list(1, 2, NULL)
)
mv <- make_values(
  list("A", "B", "C"), list(NULL, NULL, NULL),
  list("valid", NULL, "invalid")
)
fit_pmml_4 <- add_data_field_children(fit_pmml,
  field = "Sepal.Length",
  interval = mi, values = mv
)
```

`add_data_field_children` *Add 'Interval' and 'Value' child elements to a given DataField element in a given PMML file.*

Description

Add 'Interval' and 'Value' child elements to a given DataField element in a given PMML file.

Usage

```
add_data_field_children(
  xml_model = NULL,
  field = NULL,
  intervals = NULL,
  values = NULL,
  namespace = "4_4",
  ...
)
```

Arguments

`xml_model`	The PMML model in a XML node format. If the model is a text file, it should be converted to an XML node, for example, using the file_to_xml_node function.
`field`	The field to which the attributes are to be added. This is used when the attributes are a vector of name-value pairs, intended for this one field.
`intervals`	The 'Interval' elements given as a list
`values`	The 'Value' elements given as a list.
`namespace`	The namespace of the PMML model. This is frequently also the PMML version of the model.
`...`	Further arguments passed to or from other methods.

Details

The PMML format allows a DataField element to have 'Interval' and 'Value' child elements which although useful, may not always be present in a PMML model. This function allows one to take an existing PMML file and add these elements to the DataFields.

The 'Interval' elements or the 'Value' elements can be typed in, but more conveniently created by using the helper functions 'make_intervals' and 'MakeValues'. This function can then add these extra information to the PMML.

Value

An object of class `XMLNode` as that defined by the **XML** package. This represents the top level, or root node, of the XML document and is of type PMML. It can be written to file with `saveXML`.

Author(s)

Tridivesh Jena

Examples

```
# Make a sample model:
fit <- lm(Sepal.Length ~ ., data = iris[, -5])
fit_pmml <- pmml(fit)

# The resulting model has data fields but with no 'Interval' or Value'
# elements. This object is already an xml node (not an external text
# file), so there is no need to convert it to an xml node object.

# Add an 'Interval' element node by typing it in
fit_pmml_2 <- add_data_field_children(fit_pmml,
  field = "Sepal.Length",
  intervals = list(newXMLNode("Interval",
    attrs = c(closure = "openClosed", rightMargin = 3)
  ))
)

# Use helper functions to create list of 'Interval' and 'Value'
# elements. We define the 3 Intervals as ,1]  (1,2)  and [2,
mi <- make_intervals(
  list("openClosed", "openOpen", "closedOpen"),
  list(NULL, 1, 2), list(1, 2, NULL)
)

# Define 3 values, none with a 'displayValue' attribute and 1 value
# defined as 'invalid'. The 2nd one is 'valid' by default.
mv <- make_values(
  list(1.1, 2.2, 3.3), list(NULL, NULL, NULL),
  list("valid", NULL, "invalid")
)

# As an example, apply these to the Sepal.Length field:
fit_pmml_3 <- add_data_field_children(fit_pmml, field = "Sepal.Length", intervals = mi, values = mv)

# Only defined 'Interval's:
fit_pmml_3 <- add_data_field_children(fit_pmml, field = "Sepal.Length", intervals = mi)
```

`add_mining_field_attributes` *Add attribute values to an existing MiningField element in a given PMML file.*

Description

Add attribute values to an existing MiningField element in a given PMML file.

Usage

```
add_mining_field_attributes(
  xml_model = NULL,
  attributes = NULL,
  namespace = "4_4",
  ...
)
```

Arguments

xml_model	The PMML model in a XML node format. If the model is a text file, it should be converted to an XML node, for example, using the file_to_xml_node function.
attributes	The attributes to be added to the mining fields. The user should make sure that the attributes being added are allowed in the PMML schema.
namespace	The namespace of the PMML model. This is frequently also the PMML version of the model.
...	Further arguments passed to or from other methods.

Details

The PMML format allows a MiningField element to have attributes 'usageType', 'missingValueReplacement' and 'invalidValueTreatment' which although useful, may not always be present in a PMML model. This function allows one to take an existing PMML file and add these attributes to the MiningFields.

The attribute information should be provided as a dataframe; each row corresponding to an attribute name and each column corresponding to a variable name. This way one can add as many attributes to as many variables as one wants in one step. On the other extreme, a one-by-one data frame may be used to add one new attribute to one variable. This function may be used multiple times to add new attribute values step-by-step. This function overwrites any pre-existing attribute values, so it must be used with care. However, this is by design as this feature is meant to help an user defined new attribute values at different times. For example, one may use this to impute missing values in a model at different times.

Value

An object of class XMLNode as that defined by the **XML** package. This represents the top level, or root node, of the XML document and is of type PMML. It can be written to file with saveXML.

Author(s)

Tridivesh Jena

Examples

```
# Make a sample model
fit <- lm(Sepal.Length ~ ., data = iris[, -5])
fit_pmml <- pmml(fit)
```

```
# The resulting model has mining fields with no information
# besides fieldName, dataType and optype. This object is
# already an xml node (not an external text file), so there
# is no need to convert it to an xml node object.

# Create data frame with attribute information:
attributes <- data.frame(
  c("active", 1.1, "asIs"),
  c("active", 2.2, "asIs"),
  c("active", NA, "asMissing"),
  stringsAsFactors = TRUE
)
rownames(attributes) <- c(
  "usageType", "missingValueReplacement",
  "invalidValueTreatment"
)
colnames(attributes) <- c(
  "Sepal.Width", "Petal.Length",
  "Petal.Width"
)

# Although not needed in this first try, necessary to easily
# add new values later:
for (k in 1:ncol(attributes)) {
  attributes[[k]] <- as.character(attributes[[k]])
}

fit_pmml <- add_mining_field_attributes(fit_pmml, attributes, namespace = "4_4")
```

`add_output_field` *Add Output nodes to a PMML object.*

Description

Add Output nodes to a PMML object.

Usage

```
add_output_field(
  xml_model = NULL,
  outputNodes = NULL,
  at = "End",
  xformText = NULL,
  nodeName = NULL,
  attributes = NULL,
  whichOutput = 1,
  namespace = "4_4"
)
```

Arguments

xml_model	The PMML model to which the OutputField elements are to be added
outputNodes	The Output nodes to be added. These may be created using the 'make_output_nodes' helper function
at	Given an Output element, the 1 based index after which the given Output child element should be inserted at
xformText	Post-processing information to be included in the OutputField element. This expression will be processed by the function_to_pmml function
nodeName	The name of the element to be added
attributes	The attributes to be added
whichOutput	The index of the Output element
namespace	The namespace of the PMML model

Details

This function is meant to add any post-processing information to an existing model via the OutputField element. One can also use this to tell the PMML model to output other values not automatically added to the model output. The first method is to use the 'make_output_nodes' helper function to make a list of output elements to be added. 'whichOutput' lets the function know which of the Output elements we want to work with; there may be more than one in a multiple model file. One can then add those elements there, at the desired index given by the 'at' parameter; the elements are inserted after the OutputField element at the 'at' index. In other words, find the 'whichOutput' Output element, add the 'outputNodes' child elements (which should be OutputField nodes) at the 'at' position in the child nodes. This function can also be used with the 'nodeName' and 'attributes' to add the list of attributes to an OutputField element with name 'nodeName' element using the 'xml_model', 'outputNodes' and 'at' parameters. Finally, one can use this to add the transformation expression given by the 'xformText' parameter to the node with name 'nodeName'. The string given via 'xformText' is converted to an XML expression similarly to the function_to_pmml function. In other words, find the OutputField node with the name 'nodeName' and add the list of attributes given with 'attributes' and also, add the child transformations given in the 'xformText' parameter.

Value

Output node with the OutputField elements inserted.

Author(s)

Tridivesh Jena

Examples

```
# Load the standard iris dataset
data(iris)

# Create a linear model and convert it to PMML
mod <- lm(Sepal.Length ~ ., iris)
```

```
pmod <- pmml(mod)

# Create additional output nodes
onodes0 <- make_output_nodes(
  name = list("OutputField", "OutputField"),
  attributes = list(list(
    name = "dbl",
    optype = "continuous"
  ), NULL),
  expression = list("ln(x)", "ln(x/(1-x))")
)
onodes2 <- make_output_nodes(
  name = list("OutputField", "OutputField"),
  attributes = list(
    list(
      name = "F1",
      dataType = "double", optype = "continuous"
    ),
    list(name = "F2")
  )
)

# Create new pmml objects with the output nodes appended
pmod2 <- add_output_field(
  xml_model = pmod, outputNodes = onodes2, at = "End",
  xformText = NULL, nodeName = NULL, attributes = NULL,
  whichOutput = 1
)
pmod2 <- add_output_field(
  xml_model = pmod, outputNodes = onodes0, at = "End",
  xformText = NULL, nodeName = NULL,
  attributes = NULL, whichOutput = 1
)

# Create nodes with attributes and transformations
pmod3 <- add_output_field(xml_model = pmod2, outputNodes = onodes2, at = 2)
pmod4 <- add_output_field(
  xml_model = pmod2, xformText = list("exp(x) && !x"),
  nodeName = "Predicted_Sepal.Length"
)

att <- list(datype = "dbl", optpe = "dsc")
pmod5 <- add_output_field(
  xml_model = pmod2, nodeName = "Predicted_Sepal.Length",
  attributes = att
)
```

---

audit *Audit: artificially constructed dataset*

Description

This is an artificial dataset consisting of fictional clients who have been audited, perhaps for tax refund compliance. For each case an outcome is recorded (whether the taxpayer's claims had to be adjusted or not) and any amount of adjustment that resulted is also recorded.

Format

A data frame containing:

Age	Numeric
Employment	Categorical string with 7 levels
Education	Categorical string with 16 levels
Marital	Categorical string with 6 levels
Occupation	Categorical string with 14 levels
Income	Numeric
Sex	Categorical string with 2 levels
Deductions	Numeric
Hours	Numeric
Accounts	Categorical string with 32 levels
Adjustment	Numeric
Adjusted	Numeric value 0 or 1

References

- Togaware rattle package : ***Audit dataset***
- DMG description of the Audit dataset

Examples

```
data(audit, package = "pmml")
```

`file_to_xml_node` *Read in a file and parse it into an object of type XMLNode.*

Description

Read in a file and parse it into an object of type XMLNode.

Usage

```
file_to_xml_node(file)
```

Arguments

`file`	The external file to be read in. This file can be any file in PMML format, regardless of the source or model type.

Details

Read in an external file and convert it into an XMLNode to be used subsequently by other R functions.

This format is the one that will be obtained when a model is constructed in R and output in PMML format.

This function is mainly meant to be used to read in external files instead of depending on models saved in R. As an example, the pmml package requires as input an object of type XMLNode before its functions can be applied. Function 'file_to_xml_node' can be used to read in an existing PMML file, convert it to an XML node and then make it available for use by any of the pmml functions.

Value

An object of class `XMLNode` as that defined by the **XML** package. This represents the top level, or root node, of the XML document and is of type PMML. It can be written to file with `saveXML`.

Author(s)

Tridivesh Jena

Examples

```
## Not run:
# Define some transformations:
iris_box <- xform_wrap(iris)
iris_box <- xform_z_score(iris_box, xform_info = "column1->d1")
iris_box <- xform_z_score(iris_box, xform_info = "column2->d2")

# Make a LocalTransformations element and save it to an external file:
pmml_trans <- pmml(NULL, transforms = iris_box)
write(toString(pmml_trans), file = "xform_iris.pmml")

# Later, we may need to read in the PMML model into R
# 'lt' below is now a XML Node, as opposed to a string:
lt <- file_to_xml_node("xform_iris.pmml")

## End(Not run)
```

---

function_to_pmml *Convert an R expression to PMML.*

Description

Convert an R expression to PMML.

Usage

```
function_to_pmml(expr)
```

Arguments

expr	An R expression enclosed in quotes.

Details

As long as the expression passed to the function is a valid R expression (e.g., no unbalanced parenthesis), it can contain arbitrary function names not defined in R. Variables in the expression passed to ‘xform_function‘ are always assumed to be fields, and not substituted. That is, even if ‘x‘ has a value in the R environment, the resulting expression will still use ‘x‘.

An expression such as ‘foo(x)‘ is treated as a function ‘foo‘ with argument ‘x‘. Consequently, passing in an R vector ‘c(1,2,3)‘ to ‘function_to_pmml()‘ will produce PMML where ‘c‘ is a function and ‘1,2,3‘ are the arguments.

An expression starting with ’-’ or ’+’ (for example, "-3" or "-(a+b)") will be treated as if there is a 0 before the initial ’-’ or ’+’ sign. This makes it possible to represent expressions that start with a sign, since PMML’s ’-’ and ’+’ functions require two arguments. The resulting PMML node will have a constant 0 as a child.

Value

PMML version of the input expression

Author(s)

Dmitriy Bolotov

Examples

```
# Operator precedence and parenthesis
func_pmml <- function_to_pmml("1 + 3/5 - (4 * 2)")

# Nested arbitrary functions
func_pmml <- function_to_pmml("foo(bar(x)) - bar(foo(y-z))")

# If-else expression
func_pmml <- function_to_pmml("if (x==3) { 3 } else { 0 }")

# If-else with boolean output
func_pmml <- function_to_pmml("if (x==3) { TRUE } else { FALSE }")

# Function with string argument types
func_pmml <- function_to_pmml("colors('red','green','blue')")

# Sign in front of expression
func_pmml <- function_to_pmml("-(x/y)")
```

houseVotes84 *Modified 1984 United States Congressional Voting Records Database*

Description

This data set includes votes for each of the U.S. House of Representatives Congressmen on the 16 key votes identified by the CQA. The CQA lists nine different types of votes: voted for, paired for, and announced for (these three simplified to yea), voted against, paired against, and announced against (these three simplified to nay), voted present, voted present to avoid conflict of interest, and did not vote or otherwise make a position known (these three simplified to an unknown disposition). Originally containing a binomial variable "class" and 16 other binary variables, those 16 variables have been renamed to simply "V1","V2",...,"V16".

Format

A data frame containing:

Class	Boolean variable
V1	Boolean variable
V2	Boolean variable
V3	Boolean variable
V4	Boolean variable
V5	Boolean variable
V6	Boolean variable
V7	Boolean variable
V8	Boolean variable
V9	Boolean variable
V10	Boolean variable
V11	Boolean variable
V12	Boolean variable
V13	Boolean variable
V14	Boolean variable
V15	Boolean variable
V16	Boolean variable

References

UCI Machine Learning Repository

Examples

```
data(houseVotes84, package = "pmml")
```

make_intervals *Create Interval elements, most likely to add to a DataDictionary element.*

Description

Create Interval elements, most likely to add to a DataDictionary element.

Usage

```
make_intervals(
  closure = NULL,
  leftMargin = NULL,
  rightMargin = NULL,
  namespace = "4_4"
)
```

Arguments

closure	The 'closure' attribute of each 'Interval' element to be created in order.
leftMargin	The 'leftMargin' attribute of each 'Interval' element to be created in order.
rightMargin	The 'rightMargin' attribute of each 'Interval' element to be created in order.
namespace	The namespace of the PMML model

Details

The 'Interval' element allows 3 attributes, all of which may be defined in the 'make_intervals' function. The value of these attributes should be provided as a list. Thus the elements of the 'leftMargin' for example define the value of that attribute for each 'Interval' element in order.

Value

PMML Intervals elements.

Author(s)

Tridivesh Jena

See Also

make_values to make Values child elements, add_data_field_children to add these xml fragments to the DataDictionary PMML element.

Examples

```
# make 3 Interval elements
# we define the 3 Intervals as ,1]  (1,2)  and [2,
mi <- make_intervals(
  list("openClosed", "openOpen", "closedOpen"),
  list(NULL, 1, 2), list(1, 2, NULL)
)
```

`make_output_nodes` *Add Output nodes to a PMML object.*

Description

Add Output nodes to a PMML object.

Usage

```
make_output_nodes(
  name = "OutputField",
  attributes = NULL,
  expression = NULL,
  namespace = "4_4"
)
```

Arguments

`name`	The name of the element to be created.
`attributes`	The node attributes to be added.
`expression`	Post-processing information to be included in the element. This expression will be processed by `function_to_pmml`.
`namespace`	The namespace of the PMML model.

Details

Create a list of nodes with names `'name'`, attributes `'attributes'` and child elements `'expression'`. `'expression'` is a string converted to XML similar to `function_to_pmml`.

Meant to create OutputField elements, 'expressions' can be used to add post-processing transformations to a model. To create multiple such nodes, all the parameters must be given as lists of equal length.

Value

List of nodes

Author(s)

Tridivesh Jena

Examples

```
# Make two nodes, one with attributes
two_nodes <- make_output_nodes(
  name = list("OutputField", "OutputField"),
  attributes = list(list(name = "dbl", optype = "continuous"), NULL),
  expression = list("ln(x)", "ln(x/(1-x))")
)
```

`make_values` *Create Values element, most likely to add to a DataDictionary element.*

Description

Create Values element, most likely to add to a DataDictionary element.

Usage

```
make_values(
  value = NULL,
  displayValue = NULL,
  property = NULL,
  namespace = "4_4"
)
```

Arguments

`value`	The 'value' attribute of each 'Value' element to be created in order.
`displayValue`	The 'displayValue' attribute of each 'Value' element to be created in order.
`property`	The 'property' attribute of each 'Value' element to be created in order.
`namespace`	The namespace of the PMML model

Details

This function is used the same way as the `make_intervals` function. If certain attributes for an element should not be included, they should be input in the list as NULL.

Value

PMML Values elements.

Author(s)

Tridivesh Jena

See Also

`make_intervals` to make Interval child elements, `add_data_field_children` to add these xml fragments to the DataDictionary PMML element.

Examples

```
# define 3 values, none with a 'displayValue' attribute and 1 value
# defined as 'invalid'. The 2nd one is 'valid' by default.
mv <- make_values(
  list(1.1, 2.2, 3.3), list(NULL, NULL, NULL),
  list("valid", NULL, "invalid")
)
```

---

pmml *Generate the PMML representation for R objects.*

Description

pmml is a generic function implementing S3 methods used to produce the PMML (Predictive Model Markup Language) representation of an R model. The resulting PMML file can then be imported into other systems that accept PMML.

Usage

```
pmml(
  model = NULL,
  model_name = "R_Model",
  app_name = "SoftwareAG PMML Generator",
  description = NULL,
  copyright = NULL,
  model_version = NULL,
  transforms = NULL,
  ...
)
```

Arguments

model	An object to be converted to PMML.
model_name	A name to be given to the PMML model.
app_name	The name of the application that generated the PMML.
description	A descriptive text for the Header element of the PMML.
copyright	The copyright notice for the model.
model_version	A string specifying the model version.
transforms	Data transformations.
...	Further arguments passed to or from other methods.

Details

The data transformation functions previously available in the separate `pmmlTransformations` package have been merged into `pmml` starting with version 2.0.0.

This function can also be used to output variable transformations in PMML format. In particular, it can be used as a transformations generator. Various transformation operations can be implemented in R and those transformations can then be output in PMML format by calling the function with a NULL value for the model input and a data transformation object as the transforms input. Please see the documentation for `xform_wrap` for more information on how to create a data transformation object.

In addition, the `pmml` function can also be called using a pre-existing PMML model as the first input and a data transformation object as the transforms input. The result is a new PMML model with the transformation inserted as a "LocalTransformations" element in the original model. If the original model already had a "LocalTransformations" element, the new information will be appended to that element. If the model variables are derived directly from a chain of transformations defined in the transforms input, the field names in the model are replaced with the original field names with the correct data types to make a consistent model. The covered cases include model fields derived from an original field, model fields derived from a chain of transformations starting from an original field and multiple fields derived from the same original field.

This package exports models to PMML version 4.4.1.

Please note that package **XML_3.95-0.1** or later is required to perform the full and correct functionality of **pmml**.

If data used for an R model contains features of type `character`, these must be converted to factors before the model is trained and converted with `pmml`.

A list of all the supported models and packages is available in the vignette:

`vignette("packages_and_functions", package="pmml")`.

Value

An object of class `XMLNode` as that defined by the **XML** package. This represents the top level, or root node, of the XML document and is of type PMML. It can be written to file with `saveXML`.

Author(s)

Graham Williams

References

- PMML home page
- PMML transformations

See Also

`pmml.ada`, `pmml.rules`, `pmml.coxph`, `pmml.cv.glmnet`, `pmml.glm`, `pmml.hclust`, `pmml.kmeans`, `pmml.ksvm`, `pmml.lm`, `pmml.multinom`, `pmml.naiveBayes`, `pmml.neighbr`, `pmml.nnet`, `pmml.rpart`, `pmml.svm`, `pmml.xgb.Booster`

Examples

```
# Build an lm model
iris_lm <- lm(Sepal.Length ~ ., data = iris)

# Convert to pmml
iris_lm_pmml <- pmml(iris_lm)

# Create a data transformation object
iris_trans <- xform_wrap(iris)

# Transform the 'Sepal.Length' variable
iris_trans <- xform_min_max(iris_trans, xform_info = "column1->d_sl")

# Output the tranformation in PMML format
iris_trans_pmml <- pmml(NULL, transforms = iris_trans)
```

---

pmml.ada *Generate the PMML representation for an ada object from the package* **ada**.

Description

Generate the PMML representation for an ada object from the package **ada**.

Usage

```
## S3 method for class 'ada'
pmml(
  model,
  model_name = "AdaBoost_Model",
  app_name = "SoftwareAG PMML Generator",
  description = "AdaBoost Model",
  copyright = NULL,
  model_version = NULL,
  transforms = NULL,
  missing_value_replacement = NULL,
  ...
)
```

Arguments

model	An ada object.
model_name	A name to be given to the PMML model.
app_name	The name of the application that generated the PMML.
description	A descriptive text for the Header element of the PMML.
copyright	The copyright notice for the model.

model_version	A string specifying the model version.
transforms	Data transformations.
missing_value_replacement	Value to be used as the 'missingValueReplacement' attribute for all MiningFields.
...	Further arguments passed to or from other methods.

Details

Export the ada model in the PMML MiningModel (multiple models) format. The MiningModel element consists of a list of TreeModel elements, one in each model segment.

This function implements the discrete adaboost algorithm only. Note that each segment tree is a classification model, returning either -1 or 1. However the MiningModel (ada algorithm) is doing a weighted sum of the returned value, -1 or 1. So the value of attribute functionName of element MiningModel is set to "regression"; the value of attribute functionName of each segment tree is also set to "regression" (they have to be the same as the parent MiningModel per PMML schema). Although each segment/tree is being named a "regression" tree, the actual returned score can only be -1 or 1, which practically turns each segment into a classification tree.

The model in PMML format has 5 different outputs. The "rawValue" output is the value of the model expressed as a tree model. The boosted tree model uses a transformation of this value, this is the "boostValue" output. The last 3 outputs are the predicted class and the probabilities of each of the 2 classes (The ada package Boosted Tree models can only handle binary classification models).

Author(s)

Wen Lin

References

ada: an R package for stochastic boosting (on CRAN)

Examples

```
## Not run:
library(ada)
data(audit)

fit <- ada(Adjusted ~ Employment + Education + Hours + Income, iter = 3, audit)
fit_pmml <- pmml(fit)

## End(Not run)
```

pmml.ARIMA *Generate PMML for an ARIMA object the **forecast** package.*

Description

Generate PMML for an ARIMA object the **forecast** package.

Usage

```
## S3 method for class 'ARIMA'
pmml(
  model,
  model_name = "ARIMA_model",
  app_name = "SoftwareAG PMML Generator",
  description = "ARIMA Time Series Model",
  copyright = NULL,
  model_version = NULL,
  transforms = NULL,
  missing_value_replacement = NULL,
  ts_type = "statespace",
  cpi_levels = c(80, 95),
  ...
)
```

Arguments

model	An ARIMA object from the package **forecast**.
model_name	A name to be given to the PMML model.
app_name	The name of the application that generated the PMML.
description	A descriptive text for the Header element of the PMML.
copyright	The copyright notice for the model.
model_version	A string specifying the model version.
transforms	Data transformations.
missing_value_replacement	Value to be used as the 'missingValueReplacement' attribute for all MiningFields.
ts_type	The type of time series representation for PMML: "arima" or "statespace".
cpi_levels	Vector of confidence levels for prediction intervals.
...	Further arguments passed to or from other methods.

Details

The model is represented as a PMML TimeSeriesModel.

When `ts_type = "statespace"` (by default), the R object is exported as StateSpaceModel in PMML.

When `ts_type = "arima"`, the R object is exported as ARIMA in PMML with conditional least squares (CLS). Note that ARIMA models in R are estimated using a state space representation. Therefore, when using CLS with seasonal models, forecast results between R and PMML may not match exactly. Additionally, when ts_type="arima", prediction intervals are exported for non-seasonal models only. For ARIMA models with d=2, the prediction intervals between R and PMML may not match.

OutputField elements are exported with dataType "string", and contain a collection of all values up to and including the steps-ahead value supplied during scoring. String output in this form is facilitated by Extension elements in the PMML file, and is supported by Zementis Server since version 10.6.0.0.

`cpi_levels` behaves similar to `levels` in `forecast::forecast`: values must be between 0 and 100, non-inclusive.

Models with a drift term will be supported in a future version.

Transforms are currently not supported for ARIMA models.

Value

PMML representation of the `ARIMA` object.

Author(s)

Dmitriy Bolotov

Examples

```
## Not run:
library(forecast)

# non-seasonal model
data("WWWusage")
mod <- Arima(WWWusage, order = c(3, 1, 1))
mod_pmml <- pmml(mod)

# seasonal model
data("JohnsonJohnson")
mod_02 <- Arima(JohnsonJohnson,
  order = c(1, 1, 1),
  seasonal = c(1, 1, 1)
)
mod_02_pmml <- pmml(mod_02)

# non-seasonal model exported with Conditional Least Squares
data("WWWusage")
mod <- Arima(WWWusage, order = c(3, 1, 1))
```

```
mod_pmml <- pmml(mod, ts_type = "arima")

## End(Not run)
```

`pmml.coxph` *Generate the PMML representation for a coxph object from the package* **survival**.

Description

Generate the PMML representation for a coxph object from the package **survival**.

Usage

```
## S3 method for class 'coxph'
pmml(
  model,
  model_name = "CoxPH_Survival_Regression_Model",
  app_name = "SoftwareAG PMML Generator",
  description = "CoxPH Survival Regression Model",
  copyright = NULL,
  model_version = NULL,
  transforms = NULL,
  missing_value_replacement = NULL,
  ...
)
```

Arguments

`model`	A coxph object.
`model_name`	A name to be given to the PMML model.
`app_name`	The name of the application that generated the PMML.
`description`	A descriptive text for the Header element of the PMML.
`copyright`	The copyright notice for the model.
`model_version`	A string specifying the model version.
`transforms`	Data transformations.
`missing_value_replacement`	Value to be used as the 'missingValueReplacement' attribute for all MiningFields.
`...`	Further arguments passed to or from other methods.

Details

A coxph object is the result of fitting a proportional hazards regression model, using the coxph function from the package **survival**. Although the **survival** package supports special terms "cluster", "tt" and "strata", only the special term "strata" is supported by the **pmml** package. Note that special term "strata" cannot be a multiplicative variable and only numeric risk regression is supported.

Author(s)

Graham Williams

References

coxph: Survival Analysis

`pmml.cv.glmnet` *Generate the PMML representation for a cv.glmnet object from the package* **glmnet**.

Description

Generate the PMML representation for a cv.glmnet object from the package **glmnet**.

Usage

```
## S3 method for class 'cv.glmnet'
pmml(
  model,
  model_name = "Elasticnet_Model",
  app_name = "SoftwareAG PMML Generator",
  description = "Generalized Linear Regression Model",
  copyright = NULL,
  model_version = NULL,
  transforms = NULL,
  missing_value_replacement = NULL,
  dataset = NULL,
  s = NULL,
  ...
)
```

Arguments

`model`	A cv.glmnet object.
`model_name`	A name to be given to the PMML model.
`app_name`	The name of the application that generated the PMML.
`description`	A descriptive text for the Header element of the PMML.
`copyright`	The copyright notice for the model.
`model_version`	A string specifying the model version.
`transforms`	Data transformations.
`missing_value_replacement`	Value to be used as the 'missingValueReplacement' attribute for all MiningFields.
`dataset`	Data used to train the cv.glmnet model.

s	'lambda' parameter at which to output the model. If not given, the lambda.1se parameter from the model is used instead.
...	Further arguments passed to or from other methods.

Details

The glmnet package expects the input and predicted values in a matrix format - not as arrays or data frames. As of now, it will also accept numerical values only. As such, any string variables must be converted to numerical ones. One possible way to do so is to use data transformation functions from this package. However, the result is a data frame. In all cases, lists, arrays and data frames can be converted to a matrix format using the data.matrix function from the base package. Given a data frame df, a matrix m can thus be created by using `m <- data.matrix(df)`.

The PMML language requires variable names which will be read in as the column names of the input matrix. If the matrix does not have variable names, they will be given the default values of "X1", "X2", ...

Currently, only gaussian and poisson family types are supported.

Value

PMML representation of the cv.glmnet object.

Author(s)

Tridivesh Jena

References

glmnet: Lasso and elastic-net regularized generalized linear models (on CRAN)

Examples

```
## Not run:
library(glmnet)

# Create a simple predictor (x) and response(y) matrices:
x <- matrix(rnorm(100 * 20), 100, 20)
y <- rnorm(100)

# Build a simple gaussian model:
model1 <- cv.glmnet(x, y)

# Output the model in PMML format:
model1_pmml <- pmml(model1)

# Shift y between 0 and 1 to create a poisson response:
y <- y - min(y)

# Give the predictor variables names (default values are V1,V2,...):
name <- NULL
for (i in 1:20) {
```

```
  name <- c(name, paste("variable", i, sep = ""))
}
colnames(x) <- name

# Create a simple poisson model:
model2 <- cv.glmnet(x, y, family = "poisson")

# Output the regression model in PMML format at the lambda
# parameter = 0.006:
model2_pmml <- pmml(model2, s = 0.006)

## End(Not run)
```

---

pmml.gbm *Generate the PMML representation for a gbm object from the package* **gbm**.

Description

Generate the PMML representation for a gbm object from the package **gbm**.

Usage

```
## S3 method for class 'gbm'
pmml(
  model,
  model_name = "GBM_Model",
  app_name = "SoftwareAG PMML Generator",
  description = "Generalized Boosted Tree Model",
  copyright = NULL,
  model_version = NULL,
  transforms = NULL,
  missing_value_replacement = NULL,
  ...
)
```

Arguments

model	A gbm object.
model_name	A name to be given to the PMML model.
app_name	The name of the application that generated the PMML.
description	A descriptive text for the Header element of the PMML.
copyright	The copyright notice for the model.
model_version	A string specifying the model version.
transforms	Data transformations.

`missing_value_replacement`
: Value to be used as the 'missingValueReplacement' attribute for all MiningFields.

`...`
: Further arguments passed to or from other methods.

Details

The 'gbm' function uses various distribution types to fit a model; currently only the "bernoulli", "poisson" and "multinomial" distribution types are supported.

For all cases, the model output includes the gbm prediction type "link" and "response".

Value

PMML representation of the gbm object.

Author(s)

Tridivesh Jena

References

gbm: Generalized Boosted Regression Models (on CRAN)

Examples

```
## Not run:
library(gbm)
data(audit)

mod <- gbm(Adjusted ~ .,
  data = audit[, -c(1, 4, 6, 9, 10, 11, 12)],
  n.trees = 3, interaction.depth = 4
)

mod_pmml <- pmml(mod)

# Classification example:
mod2 <- gbm(Species ~ .,
  data = iris, n.trees = 2,
  interaction.depth = 3, distribution = "multinomial"
)

# The PMML will include a regression model to read the gbm object outputs
# and convert to a "response" prediction type.
mod2_pmml <- pmml(mod2)

## End(Not run)
```

pmml.glm *Generate the PMML representation for a glm object from the package* **stats**.

Description

Generate the PMML representation for a glm object from the package **stats**.

Usage

```
## S3 method for class 'glm'
pmml(
  model,
  model_name = "General_Regression_Model",
  app_name = "SoftwareAG PMML Generator",
  description = "Generalized Linear Regression Model",
  copyright = NULL,
  model_version = NULL,
  transforms = NULL,
  missing_value_replacement = NULL,
  weights = NULL,
  ...
)
```

Arguments

model	A glm object.
model_name	A name to be given to the PMML model.
app_name	The name of the application that generated the PMML.
description	A descriptive text for the Header element of the PMML.
copyright	The copyright notice for the model.
model_version	A string specifying the model version.
transforms	Data transformations.
missing_value_replacement	Value to be used as the 'missingValueReplacement' attribute for all MiningFields.
weights	The weights used for building the model.
...	Further arguments passed to or from other methods.

Details

The function exports the glm model in the PMML GeneralRegressionModel format.

Note on glm models for 2-class problems: a dataset where the target categorical variable has more than 2 classes may be turned into a 2-class problem by creating a new target variable that is TRUE

for a particular class and FALSE for all other classes. While the R formula function allows such a transformation to be passed directly to it, this may cause issues when the model is converted to PMML. Therefore, it is advised to create a new 2-class separately, and then pass that variable to glm(). This is shown in an example below.

Value

PMML representation of the glm object.

References

R project: Fitting Generalized Linear Models

Examples

```
## Not run:
data(iris)
mod <- glm(Sepal.Length ~ ., data = iris, family = "gaussian")
mod_pmml <- pmml(mod)
rm(mod, mod_pmml)

data(audit)
mod <- glm(Adjusted ~ Age + Employment + Education + Income, data = audit, family = binomial(logit))
mod_pmml <- pmml(mod)
rm(mod, mod_pmml)

# Create a new 2-class target from a 3-class variable:
data(iris)
dat <- iris[, 1:4]
# Add a new 2-class target "Species_setosa" before passing it to glm():
dat$Species_setosa <- iris$Species == "setosa"
mod <- glm(Species_setosa ~ ., data = dat, family = binomial(logit))
mod_pmml <- pmml(mod)
rm(dat, mod, mod_pmml)

## End(Not run)
```

---

pmml.hclust *Generate the PMML representation for a hclust object from the package* **amap**.

Description

Generate the PMML representation for a hclust object from the package **amap**.

Usage

```
## S3 method for class 'hclust'
pmml(
  model,
  model_name = "HClust_Model",
  app_name = "SoftwareAG PMML Generator",
  description = "Hierarchical Cluster Model",
  copyright = NULL,
  model_version = NULL,
  transforms = NULL,
  missing_value_replacement = NULL,
  centers,
  ...
)
```

Arguments

model	A hclust object.
model_name	A name to be given to the PMML model.
app_name	The name of the application that generated the PMML.
description	A descriptive text for the Header element of the PMML.
copyright	The copyright notice for the model.
model_version	A string specifying the model version.
transforms	Data transformations.
missing_value_replacement	Value to be used as the 'missingValueReplacement' attribute for all MiningFields.
centers	A list of means to represent the clusters.
...	Further arguments passed to or from other methods.

Details

This function converts a hclust object created by the `hclusterpar` function from the **amap** package. A `hclust` object is a cluster model created hierarchically. The data is divided recursively until a criteria is met. This function then takes the final model and represents it as a standard k-means cluster model. This is possible since while the method of constructing the model is different, the final model can be represented in the same way.

To use this pmml function, therefore, one must pick the number of clusters desired and the coordinate values at those cluster centers. This can be done using the `hclusterpar` and `centers.hclust` functions from the **amap** and `rattle` packages respectively.

The hclust object will be approximated by k centroids and is converted into a PMML representation for kmeans clusters.

Value

PMML representation of the hclust object.

Author(s)

Graham Williams

References

R project: Hierarchical Clustering

Examples

```
## Not run:

# Cluster the 4 numeric variables of the iris dataset.
library(amap)
library(rattle)

model <- hclusterpar(iris[, -5])

# Get the information about the cluster centers. The last
# parameter of the function used is the number of clusters
# desired.
centerInfo <- centers.hclust(iris[, -5], model, 3)

# Convert to pmml
model_pmml <- pmml(model, centers = centerInfo)

## End(Not run)
```

`pmml.iForest` *Generate PMML for an iForest object from the* ***isofor*** *package.*

Description

Generate PMML for an iForest object from the **isofor** package.

Usage

```
## S3 method for class 'iForest'
pmml(
  model,
  model_name = "isolationForest_Model",
  app_name = "SoftwareAG PMML Generator",
  description = "Isolation Forest Model",
  copyright = NULL,
  model_version = NULL,
  transforms = NULL,
  missing_value_replacement = NULL,
  anomaly_threshold = 0.6,
  parent_invalid_value_treatment = "returnInvalid",
```

```
  child_invalid_value_treatment = "asIs",
  ...
)
```

Arguments

`model`	An iForest object from package **isofor**.
`model_name`	A name to be given to the PMML model.
`app_name`	The name of the application that generated the PMML.
`description`	A descriptive text for the Header element of the PMML.
`copyright`	The copyright notice for the model.
`model_version`	A string specifying the model version.
`transforms`	Data transformations.
`missing_value_replacement`	Value to be used as the 'missingValueReplacement' attribute for all MiningFields.
`anomaly_threshold`	Double between 0 and 1. Predicted values greater than this are classified as anomalies.
`parent_invalid_value_treatment`	Invalid value treatment at the top MiningField level.
`child_invalid_value_treatment`	Invalid value treatment at the model segment MiningField level.
`...`	Further arguments passed to or from other methods.

Details

This function converts the iForest model object to the PMML format. The PMML outputs the anomaly score as well as a boolean value indicating whether the input is an anomaly or not. This is done by simply comparing the anomaly score with `anomaly_threshold`, a parameter in the `pmml` function. The iForest function automatically adds an extra level to all categorical variables, labelled "."; this is kept in the PMML representation even though the use of this extra factor in the predict function is unclear.

Value

PMML representation of the `iForest` object.

Author(s)

Tridivesh Jena

References

isofor package on GitHub

See Also

pmml

Examples

```
## Not run:

# Build iForest model using iris dataset. Create an isolation
# forest with 10 trees. Sample 30 data points at a time from
# the iris dataset to fit the trees.
library(isofor)
data(iris)
mod <- iForest(iris, nt = 10, phi = 30)

# Convert to PMML:
mod_pmml <- pmml(mod)

## End(Not run)
```

---

pmml.kmeans *Generate the PMML representation for a kmeans object from the package* **stats**.

Description

The kmeans object (a cluster described by k centroids) is converted into a PMML representation.

Usage

```
## S3 method for class 'kmeans'
pmml(
  model,
  model_name = "KMeans_Model",
  app_name = "SoftwareAG PMML Generator",
  description = "KMeans cluster model",
  copyright = NULL,
  model_version = NULL,
  transforms = NULL,
  missing_value_replacement = NULL,
  algorithm_name = "KMeans: Hartigan and Wong",
  ...
)
```

Arguments

model	A kmeans object.
model_name	A name to be given to the PMML model.
app_name	The name of the application that generated the PMML.
description	A descriptive text for the Header element of the PMML.
copyright	The copyright notice for the model.
model_version	A string specifying the model version.
transforms	Data transformations.
missing_value_replacement	Value to be used as the 'missingValueReplacement' attribute for all MiningFields.
algorithm_name	The variety of kmeans used.
...	Further arguments passed to or from other methods.

Details

A kmeans object is obtained by applying the `kmeans` function from the `stats` package. This method typically requires the user to normalize all the variables; these operations can be done using transforms so that the normalization information is included in PMML.

Author(s)

Graham Williams

References

R project: K-Means Clustering

Examples

```
## Not run:
ds <- rbind(
  matrix(rnorm(100, sd = 0.3), ncol = 2),
  matrix(rnorm(100, mean = 1, sd = 0.3), ncol = 2)
)
colnames(ds) <- c("Dimension1", "Dimension2")
cl <- kmeans(ds, 2)
cl_pmml <- pmml(cl)

## End(Not run)
```

pmml.ksvm *Generate the PMML representation for a ksvm object from the package* **kernlab**.

Description

Generate the PMML representation for a ksvm object from the package **kernlab**.

Usage

```
## S3 method for class 'ksvm'
pmml(
  model,
  model_name = "SVM_model",
  app_name = "SoftwareAG PMML Generator",
  description = "Support Vector Machine Model",
  copyright = NULL,
  model_version = NULL,
  transforms = NULL,
  missing_value_replacement = NULL,
  dataset = NULL,
  ...
)
```

Arguments

model	A ksvm object.
model_name	A name to be given to the PMML model.
app_name	The name of the application that generated the PMML.
description	A descriptive text for the Header element of the PMML.
copyright	The copyright notice for the model.
model_version	A string specifying the model version.
transforms	Data transformations.
missing_value_replacement	Value to be used as the 'missingValueReplacement' attribute for all MiningFields.
dataset	Data used to train the ksvm model.
...	Further arguments passed to or from other methods.

Details

Both classification (multi-class and binary) as well as regression cases are supported.

The following ksvm kernels are currently supported: rbfdot, polydot, vanilladot, tanhdot.

The argument dataset is required since the ksvm object does not contain information about the used categorical variable.

Value

PMML representation of the ksvm object.

References

kernlab: Kernel-based Machine Learning Lab (on CRAN)

Examples

```
## Not run:
# Train a support vector machine to perform classification.
library(kernlab)

model <- ksvm(Species ~ ., data = iris)

model_pmml <- pmml(model, dataset = iris)

## End(Not run)
```

---

pmml.lm *Generate the PMML representation for an lm object from the package* **stats**.

Description

Generate the PMML representation for an lm object from the package **stats**.

Usage

```
## S3 method for class 'lm'
pmml(
  model,
  model_name = "lm_Model",
  app_name = "SoftwareAG PMML Generator",
  description = "Linear Regression Model",
  copyright = NULL,
  model_version = NULL,
  transforms = NULL,
  missing_value_replacement = NULL,
  weights = NULL,
  ...
)
```

Arguments

model	An lm object.
model_name	A name to be given to the PMML model.

app_name	The name of the application that generated the PMML.
description	A descriptive text for the Header element of the PMML.
copyright	The copyright notice for the model.
model_version	A string specifying the model version.
transforms	Data transformations.
missing_value_replacement	Value to be used as the 'missingValueReplacement' attribute for all MiningFields.
weights	The weights used for building the model.
...	Further arguments passed to or from other methods.

Details

The resulting PMML representation will not encode interaction terms. Currently, only numeric regression is supported.

Value

PMML representation of the lm object.

Author(s)

Rajarshi Guha

References

R project: Fitting Linear Models

Examples

```
## Not run:
fit <- lm(Sepal.Length ~ ., data = iris)
fit_pmml <- pmml(fit)

## End(Not run)
```

---

pmml.multinom *Generate the PMML representation for a multinom object from package* **nnet**.

Description

Generate the multinomial logistic model in the PMML RegressionModel format. The function implements the use of numerical, categorical and multiplicative terms involving both numerical and categorical variables.

Usage

```
## S3 method for class 'multinom'
pmml(
  model,
  model_name = "multinom_Model",
  app_name = "SoftwareAG PMML Generator",
  description = "Multinomial Logistic Model",
  copyright = NULL,
  model_version = NULL,
  transforms = NULL,
  missing_value_replacement = NULL,
  ...
)
```

Arguments

model	A multinom object.
model_name	A name to be given to the PMML model.
app_name	The name of the application that generated the PMML.
description	A descriptive text for the Header element of the PMML.
copyright	The copyright notice for the model.
model_version	A string specifying the model version.
transforms	Data transformations.
missing_value_replacement	Value to be used as the 'missingValueReplacement' attribute for all MiningFields.
...	Further arguments passed to or from other methods.

Value

PMML representation of the multinom object.

Author(s)

Tridivesh Jena

References

nnet: Feed-forward Neural Networks and Multinomial Log-Linear Models (on CRAN)

Examples

```
## Not run:
library(nnet)
fit <- multinom(Species ~ ., data = iris)
fit_pmml <- pmml(fit)

## End(Not run)
```

`pmml.naiveBayes` *Generate the PMML representation for a naiveBayes object from the package* **e1071**.

Description

Generate the PMML representation for a naiveBayes object from the package **e1071**.

Usage

```
## S3 method for class 'naiveBayes'
pmml(
  model,
  model_name = "naiveBayes_Model",
  app_name = "SoftwareAG PMML Generator",
  description = "NaiveBayes Model",
  copyright = NULL,
  model_version = NULL,
  transforms = NULL,
  missing_value_replacement = NULL,
  predicted_field,
  ...
)
```

Arguments

`model`	A naiveBayes object.
`model_name`	A name to be given to the PMML model.
`app_name`	The name of the application that generated the PMML.
`description`	A descriptive text for the Header element of the PMML.
`copyright`	The copyright notice for the model.
`model_version`	A string specifying the model version.
`transforms`	Data transformations.
`missing_value_replacement`	Value to be used as the 'missingValueReplacement' attribute for all MiningFields.
`predicted_field`	Required parameter; the name of the predicted field.
`...`	Further arguments passed to or from other methods.

Details

The PMML representation of the NaiveBayes model implements the definition as specified by the Data Mining Group: intermediate probability values which are less than the threshold value are replaced by the threshold value. This is different from the prediction function of the **e1071** in which

only probability values of 0 and standard deviations of continuous variables of with the value 0 are replaced by the threshold value. The two values will therefore not match exactly for cases involving very small probabilities.

Value

PMML representation of the naiveBayes object.

Author(s)

Tridivesh Jena

References

- e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien (on CRAN)
- A. Guazzelli, T. Jena, W. Lin, M. Zeller (2013). Extending the Naive Bayes Model Element in PMML: Adding Support for Continuous Input Variables. In *Proceedings of the 19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Examples

```
## Not run:
library(e1071)

data(houseVotes84)
house <- na.omit(houseVotes84)

model <- naiveBayes(Class ~ V1 + V2 + V3, data = house, threshold = 0.003)

model_pmml <- pmml(model, dataset = house, predicted_field = "Class")

## End(Not run)
```

---

pmml.neighbr *Generate PMML for a neighbr object from the **neighbr** package.*

Description

Generate PMML for a neighbr object from the **neighbr** package.

Usage

```
## S3 method for class 'neighbr'
pmml(
  model,
  model_name = "kNN_model",
  app_name = "SoftwareAG PMML Generator",
```

```
  description = "K Nearest Neighbors Model",
  copyright = NULL,
  model_version = NULL,
  transforms = NULL,
  missing_value_replacement = NULL,
  ...
)
```

Arguments

model	A neighbr object.
model_name	A name to be given to the PMML model.
app_name	The name of the application that generated the PMML.
description	A descriptive text for the Header element of the PMML.
copyright	The copyright notice for the model.
model_version	A string specifying the model version.
transforms	Data transformations.
missing_value_replacement	Value to be used as the 'missingValueReplacement' attribute for all MiningFields.
...	Further arguments passed to or from other methods.

Details

The model is represented in the PMML NearestNeighborModel format.

The current version of this converter does not support transformations (`transforms` must be left as `NULL`), sets `categoricalScoringMethod` to "majorityVote", sets `continuousScoringMethod` to "average", and `isTransoformed` to "false".

Value

PMML representation of the neighbr object.

See Also

`pmml`, PMML KNN specification

Examples

```
## Not run:

# Continuous features with continuous target, categorical target,
# and neighbor ranking:

library(neighbr)
data(iris)

# Add an ID column to the data for neighbor ranking:
```

```
iris$ID <- c(1:150)

# Train set contains all predicted variables, features, and ID column:
train_set <- iris[1:140, ]

# Omit predicted variables and ID column from test set:
test_set <- iris[141:150, -c(4, 5, 6)]

fit <- knn(
  train_set = train_set, test_set = test_set,
  k = 3,
  categorical_target = "Species",
  continuous_target = "Petal.Width",
  comparison_measure = "squared_euclidean",
  return_ranked_neighbors = 3,
  id = "ID"
)

fit_pmml <- pmml(fit)


# Logical features with categorical target and neighbor ranking:

library(neighbr)
data("houseVotes84")

# Remove any rows with N/A elements:
dat <- houseVotes84[complete.cases(houseVotes84), ]

# Change all {yes,no} factors to {0,1}:
feature_names <- names(dat)[!names(dat) %in% c("Class", "ID")]
for (n in feature_names) {
  levels(dat[, n])[levels(dat[, n]) == "n"] <- 0
  levels(dat[, n])[levels(dat[, n]) == "y"] <- 1
}

# Change factors to numeric:
for (n in feature_names) {
  dat[, n] <- as.numeric(levels(dat[, n]))[dat[, n]]
}

# Add an ID column for neighbor ranking:
dat$ID <- c(1:nrow(dat))

# Train set contains features, predicted variable, and ID:
train_set <- dat[1:225, ]

# Test set contains features only:
test_set <- dat[226:232, !names(dat) %in% c("Class", "ID")]

fit <- knn(
  train_set = train_set, test_set = test_set,
  k = 5,
```

```
  categorical_target = "Class",
  comparison_measure = "jaccard",
  return_ranked_neighbors = 3,
  id = "ID"
)

fit_pmml <- pmml(fit)

## End(Not run)
```

---

pmml.nnet *Generate the PMML representation for a nnet object from package* **nnet**.

Description

Generate the PMML representation for a nnet object from package **nnet**.

Usage

```
## S3 method for class 'nnet'
pmml(
  model,
  model_name = "NeuralNet_model",
  app_name = "SoftwareAG PMML Generator",
  description = "Neural Network Model",
  copyright = NULL,
  model_version = NULL,
  transforms = NULL,
  missing_value_replacement = NULL,
  ...
)
```

Arguments

model	A nnet object.
model_name	A name to be given to the PMML model.
app_name	The name of the application that generated the PMML.
description	A descriptive text for the Header element of the PMML.
copyright	The copyright notice for the model.
model_version	A string specifying the model version.
transforms	Data transformations.
missing_value_replacement	Value to be used as the 'missingValueReplacement' attribute for all MiningFields.
...	Further arguments passed to or from other methods.

Details

This function supports both regression and classification neural network models. The model is represented in the PMML NeuralNetwork format.

Value

PMML representation of the nnet object.

Author(s)

Tridivesh Jena

References

nnet: Feed-forward Neural Networks and Multinomial Log-Linear Models (on CRAN)

Examples

```
## Not run:
library(nnet)
fit <- nnet(Species ~ ., data = iris, size = 4)
fit_pmml <- pmml(fit)

rm(fit)

## End(Not run)
```

---

pmml.randomForest *Generate the PMML representation for a randomForest object from the package* **randomForest**.

Description

Generate the PMML representation for a randomForest object from the package **randomForest**.

Usage

```
## S3 method for class 'randomForest'
pmml(
  model,
  model_name = "randomForest_Model",
  app_name = "SoftwareAG PMML Generator",
  description = "Random Forest Tree Model",
  copyright = NULL,
  model_version = NULL,
  transforms = NULL,
  missing_value_replacement = NULL,
  parent_invalid_value_treatment = "returnInvalid",
```

```
  child_invalid_value_treatment = "asIs",
  ...
)
```

Arguments

model	A randomForest object.
model_name	A name to be given to the PMML model.
app_name	The name of the application that generated the PMML.
description	A descriptive text for the Header element of the PMML.
copyright	The copyright notice for the model.
model_version	A string specifying the model version.
transforms	Data transformations.
missing_value_replacement	Value to be used as the 'missingValueReplacement' attribute for all MiningFields.
parent_invalid_value_treatment	Invalid value treatment at the top MiningField level.
child_invalid_value_treatment	Invalid value treatment at the model segment MiningField level.
...	Further arguments passed to or from other methods.

Details

This function outputs a Random Forest in PMML format.

Value

PMML representation of the randomForest object.

Author(s)

Tridivesh Jena

References

randomForest: Breiman and Cutler's random forests for classification and regression

Examples

```
## Not run:
# Build a randomForest model
library(randomForest)
iris_rf <- randomForest(Species ~ ., data = iris, ntree = 20)

# Convert to pmml
iris_rf_pmml <- pmml(iris_rf)
```

```
rm(iris_rf)

## End(Not run)
```

`pmml.rpart` *Generate the PMML representation for an rpart object from the package* **rpart**.

Description

Generate the PMML representation for an rpart object from the package **rpart**.

Usage

```
## S3 method for class 'rpart'
pmml(
  model,
  model_name = "RPart_Model",
  app_name = "SoftwareAG PMML Generator",
  description = "RPart Decision Tree Model",
  copyright = NULL,
  model_version = NULL,
  transforms = NULL,
  missing_value_replacement = NULL,
  dataset = NULL,
  ...
)
```

Arguments

`model`	An rpart object.
`model_name`	A name to be given to the PMML model.
`app_name`	The name of the application that generated the PMML.
`description`	A descriptive text for the Header element of the PMML.
`copyright`	The copyright notice for the model.
`model_version`	A string specifying the model version.
`transforms`	Data transformations.
`missing_value_replacement`	Value to be used as the 'missingValueReplacement' attribute for all MiningFields.
`dataset`	Data used to train the rpart model.
`...`	Further arguments passed to or from other methods.

Details

Supports regression tree as well as classification. The object is represented in the PMML TreeModel format.

Value

PMML representation of the rpart object.

Author(s)

Graham Williams

References

rpart: Recursive Partitioning (on CRAN)

Examples

```
## Not run:
library(rpart)

fit <- rpart(Species ~ ., data = iris)

fit_pmml <- pmml(fit)

## End(Not run)
```

---

pmml.rules *Generate the PMML representation for a rules or an itemset object from package* **arules**.

Description

Generate the PMML representation for a rules or an itemset object from package **arules**.

Usage

```
## S3 method for class 'rules'
pmml(
  model,
  model_name = "arules_Model",
  app_name = "SoftwareAG PMML Generator",
  description = "Association Rules Model",
  copyright = NULL,
  model_version = NULL,
  transforms = NULL,
  ...
)
```

Arguments

model	A rules or itemsets object.
model_name	A name to be given to the PMML model.
app_name	The name of the application that generated the PMML.
description	A descriptive text for the Header element of the PMML.
copyright	The copyright notice for the model.
model_version	A string specifying the model version.
transforms	Data transformations.
...	Further arguments passed to or from other methods.

Details

The model is represented in the PMML AssociationModel format.

Value

PMML representation of the rules or itemsets object.

Author(s)

Graham Williams, Michael Hahsler

References

arules: Mining Association Rules and Frequent Itemsets

---

pmml.svm *Generate the PMML representation of an svm object from the* **e1071** *package.*

Description

Generate the PMML representation of an svm object from the **e1071** package.

Usage

```
## S3 method for class 'svm'
pmml(
  model,
  model_name = "LIBSVM_Model",
  app_name = "SoftwareAG PMML Generator",
  description = "Support Vector Machine Model",
  copyright = NULL,
  model_version = NULL,
  transforms = NULL,
```

```
  missing_value_replacement = NULL,
  dataset = NULL,
  detect_anomaly = TRUE,
  ...
)
```

Arguments

model	An svm object from package **e1071**.
model_name	A name to be given to the PMML model.
app_name	The name of the application that generated the PMML.
description	A descriptive text for the Header element of the PMML.
copyright	The copyright notice for the model.
model_version	A string specifying the model version.
transforms	Data transformations.
missing_value_replacement	Value to be used as the 'missingValueReplacement' attribute for all MiningFields.
dataset	Required for one-classification only; data used to train the one-class SVM model.
detect_anomaly	Required for one-classification only; boolean indicating whether to detect anomalies (TRUE) or inliers (FALSE).
...	Further arguments passed to or from other methods.

Details

Classification and regression models are represented in the PMML SupportVectorMachineModel format. One-Classification models are represented in the PMML AnomalyDetectionModel format. Please see below for details on the differences.

Value

PMML representation of the svm object.

Classification and Regression Models

Note that the sign of the coefficient of each support vector flips between the R object and the exported PMML file for classification and regression models. This is due to the minor difference in the training/scoring formula between the LIBSVM algorithm and the DMG specification. Hence the output value of each support vector machine has a sign flip between the DMG definition and the svm prediction function.

In a classification model, even though the output of the support vector machine has a sign flip, it does not affect the final predicted category. This is because in the DMG definition, the winning category is defined as the left side of threshold 0 while the LIBSVM defines the winning category as the right side of threshold 0.

For a regression model, the exported PMML code has two OutputField elements. The OutputField `predictedValue` shows the support vector machine output per DMG definition. The OutputField `svm_predict_function` gives the value corresponding to the R predict function for the svm model. This output should be used when making model predictions.

One-Classification SVM Models

For a one-classification svm (OCSVM) model, the PMML has two OutputField elements: `anomalyScore` and one of `anomaly` or `outlier`.

The OutputField `anomalyScore` is the signed distance to the separating boundary; `anomalyScore` corresponds to the `decision.values` attribute of the output of the svm predict function in R.

The second OutputField depends the value of `detect_anomaly`. By default, `detect_anomaly` is TRUE, which results in the second OutputField being `anomaly`. The `anomaly` OutputField is TRUE when an anomaly is detected. This field conforms to the DMG definition of an anomaly detection model. This value is the opposite of the prediction by the e1071::svm object in R.

Setting `detect_anomaly` to FALSE results in the second field instead being `inlier`. This OutputField is TRUE when an inlier is detected, and conforms to the e1071 definition of one-class SVMs. This field is FALSE when an anomaly is detected; that is, the R svm model predicts whether an observation belongs to the class. When comparing the predictions from R and PMML, this field should be used, since it will match R's output.

For example, say that for an an observation, the R OCSVM model predicts a positive decision value of 0.4 and label of TRUE. According to the R object, this means that the observation is an inlier. By default, the PMML export of this model will give the following for the same input: `anomalyScore = 0.4, anomaly = "false"`. According to the PMML, the observation is not an anomaly. If the same R object is instead exported with `detect_anomaly = FALSE`, the PMML will then give: `anomalyScore = 0.4, inlier = "true"`, and this result agrees with R.

Note that there is no sign flip for `anomalyScore` between R and PMML for OCSVM models.

To export a OCSVM model, an additional argument, `dataset`, is required by the function. This argument expects a dataframe with data that was used to train the model. This is necessary because for one-class svm, the R svm object does not contain information about the data types of the features used to train the model. The exporter does not yet support the formula interface for one-classification models, so the default S3 method must be used to train the SVM. The data used to train the one-class SVM must be numeric and not of integer class.

References

* R project CRAN package: ***e1071****: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien* https://CRAN.R-project.org/package=e1071

* Chang, Chih-Chung and Lin, Chih-Jen, *LIBSVM: a library for Support Vector Machines* https://www.csie.ntu.edu.tw/~cjlin/libsvm/

See Also

pmml, PMML SVM specification

Examples

```
## Not run:
library(e1071)
data(iris)

# Classification with a polynomial kernel
fit <- svm(Species ~ ., data = iris, kernel = "polynomial")
```

```
fit_pmml <- pmml(fit)

# Regression
fit <- svm(Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width, data = iris)
fit_pmml <- pmml(fit)

# Anomaly detection with one-classification
fit <- svm(iris[, 1:4],
  y = NULL,
  type = "one-classification"
)
fit_pmml <- pmml(fit, dataset = iris[, 1:4])

# Inlier detection with one-classification
fit <- svm(iris[, 1:4],
  y = NULL,
  type = "one-classification",
  detect_anomaly = FALSE
)
fit_pmml <- pmml(fit, dataset = iris[, 1:4])

## End(Not run)
```

`pmml.xgb.Booster` *Generate PMML for a xgb.Booster object from the package* **xgboost**.

Description

Generate PMML for a xgb.Booster object from the package **xgboost**.

Usage

```
## S3 method for class 'xgb.Booster'
pmml(
  model,
  model_name = "xboost_Model",
  app_name = "SoftwareAG PMML Generator",
  description = "Extreme Gradient Boosting Model",
  copyright = NULL,
  model_version = NULL,
  transforms = NULL,
  missing_value_replacement = NULL,
  input_feature_names = NULL,
  output_label_name = NULL,
  output_categories = NULL,
  xgb_dump_file = NULL,
  parent_invalid_value_treatment = "returnInvalid",
  child_invalid_value_treatment = "asIs",
```

```
  ...
)
```

Arguments

model	An object created by the 'xgboost' function.
model_name	A name to be given to the PMML model.
app_name	The name of the application that generated the PMML.
description	A descriptive text for the Header element of the PMML.
copyright	The copyright notice for the model.
model_version	A string specifying the model version.
transforms	Data transformations.
missing_value_replacement	Value to be used as the 'missingValueReplacement' attribute for all MiningFields.
input_feature_names	Input variable names used in training the model.
output_label_name	Name of the predicted field.
output_categories	Possible values of the predicted field, for classification models.
xgb_dump_file	Name of file saved using 'xgb.dump' function.
parent_invalid_value_treatment	Invalid value treatment at the top MiningField level.
child_invalid_value_treatment	Invalid value treatment at the model segment MiningField level.
...	Further arguments passed to or from other methods.

Details

The xgboost function takes as its input either an xgb.DMatrix object or a numeric matrix. The input field information is not stored in the R model object, hence the field information must be passed on as inputs. This enables the PMML to specify field names in its model representation. The R model object does not store information about the fitted tree structure either. However, this information can be extracted from the xgb.model.dt.tree function and the file saved using the xgb.dump function. The xgboost library is therefore needed in the environment and this saved file is needed as an input as well.

The following objectives are currently supported: multi:softprob, multi:softmax, binary:logistic.

The pmml exporter will throw an error if the xgboost model model only has one tree.

The exporter only works with numeric matrices. Sparse matrices must be converted to matrix objects before training an xgboost model for the export to work correctly.

Value

PMML representation of the xgb.Booster object.

Author(s)

Tridivesh Jena

References

xgboost: Extreme Gradient Boosting

See Also

pmml, PMML schema

Examples

```
## Not run:
# Example using the xgboost package example model.

library(xgboost)
data(agaricus.train, package = "xgboost")
data(agaricus.test, package = "xgboost")

train <- agaricus.train
test <- agaricus.test

model1 <- xgboost(
  data = train$data, label = train$label,
  max_depth = 2, eta = 1, nthread = 2,
  nrounds = 2, objective = "binary:logistic"
)

# Save the tree information in an external file:
xgb.dump(model1, "model1.dumped.trees")

# Convert to PMML:
model1_pmml <- pmml(model1,
  input_feature_names = colnames(train$data),
  output_label_name = "prediction1",
  output_categories = c("0", "1"),
  xgb_dump_file = "model1.dumped.trees"
)

# Multinomial model using iris data:
model2 <- xgboost(
  data = as.matrix(iris[, 1:4]),
  label = as.numeric(iris[, 5]) - 1,
  max_depth = 2, eta = 1, nthread = 2, nrounds = 2,
  objective = "multi:softprob", num_class = 3
)

# Save the tree information in an external file:
xgb.dump(model2, "model2.dumped.trees")

# Convert to PMML:
```

```
model2_pmml <- pmml(model2,
  input_feature_names = colnames(as.matrix(iris[, 1:4])),
  output_label_name = "Species",
  output_categories = c(1, 2, 3), xgb_dump_file = "model2.dumped.trees"
)

## End(Not run)
```

`rename_wrap_var` *Rename a variable in the xform_wrap transform object.*

Description

Rename a variable in the xform_wrap transform object.

Usage

```
rename_wrap_var(wrap_object, xform_info = NA, ...)
```

Arguments

`wrap_object`	Wrapper object obtained by using the xform_wrap function on the raw data.
`xform_info`	Specification of details of the renaming.
`...`	Further arguments passed to or from other methods.

Details

Once input data is wrapped by the **xform_wrap** function, it is somewhat involved to rename a variable inside. This function makes it easier to do so. Given a variable named **input_var** and the name one wishes to rename it to, **output_var**, the rename command options are:

xform_info="input_var -> output_var"

There are two methods in which the variables can be referred to. The first method is to use its column number; given the **data** attribute of the **boxData** object, this would be the order at which the variable appears. This can be indicated in the format "column#". The second method is to refer to the variable by its name. This method will work even if the renamed value already exists; in which case there will be two variables with the same name.

If no input variable name is provided, the original object is returned with no renaming performed.

Value

R object containing the raw data, the transformed data and data statistics.

Author(s)

Tridivesh Jena

See Also

xform_wrap

Examples

```
# Load the standard iris dataset
data(iris)

# First wrap the data
iris_box <- xform_wrap(iris)

# We wish to refer to the variables "Sepal.Length" and
# "Sepal.Width" as "SL" and "SW"
iris_box <- rename_wrap_var(wrap_object = iris_box, xform_info = "column1->SL")
iris_box <- rename_wrap_var(wrap_object = iris_box, xform_info = "Sepal.Width->SW")
```

---

save_pmml *Save a pmml object as an external PMML file.*

Description

Save a pmml object to an external PMML file.

Usage

```
save_pmml(doc, name)
```

Arguments

doc	The pmml model.
name	The name of the external file where the PMML is to be saved.

Author(s)

Tridivesh Jena

Examples

```
## Not run:
# Make a gbm model:
library(gbm)
data(audit)

mod <- gbm(Adjusted ~ .,
  data = audit[, -c(1, 4, 6, 9, 10, 11, 12)],
  n.trees = 3,
  interaction.depth = 4
)
```

```
# Export to PMML:
pmod <- pmml(mod)

# Save to an external file:
save_pmml(pmod, "GBMModel.pmml")

## End(Not run)
```

`xform_discretize` *Discretize a continuous variable as indicated by interval mappings in accordance with the PMML element* ***Discretize***.

Description

Discretize a continuous variable as indicated by interval mappings in accordance with the PMML element **Discretize**.

Usage

```
xform_discretize(
  wrap_object,
  xform_info,
  table,
  default_value = NA,
  map_missing_to = NA,
  ...
)
```

Arguments

`wrap_object`	Output of xform_wrap or another transformation function.
`xform_info`	Specification of details of the transformation. This may be a name of an external file or a list of data frames. Even if only 1 variable is to be transformed, the information for that transform should be given as a list with 1 element.
`table`	Name of external CSV file containing the map from input to output values.
`default_value`	Value to be given to the transformed variable if the value of the input variable does not lie in any of the defined intervals. If 'xform_info' is a list, this is a vector with each element corresponding to the corresponding list element.
`map_missing_to`	Value to be given to the transformed variable if the value of the input variable is missing. If 'xform_info' is a list, this is a vector with each element corresponding to the corresponding list element.
`...`	Further arguments passed to or from other methods.

Details

Create a discrete variable from a continuous one as indicated by interval mappings. The discrete variable value depends on interval in which the continuous variable value lies. The mapping from intervals to discrete values can be given in an external table file referred to in the transform command or as a list of data frames.

Given a list of intervals and the discrete value each interval is linked to, a discrete variable is defined with the value indicated by the interval where it lies in. If a continuous variable **InVar** of data type **InType** is to be converted to a variable **OutVar** of data type **OutType**, the transformation command is in the format:

xform_info = "[InVar->OutVar][InType->OutType]", table="TableFileName", default_value="defVal", map_missing_to="missingVal"

where **TableFileName** is the name of the CSV file containing the interval to discrete value map. The data types of the variables can be any of the ones defined in the PMML format including integer, double or string. **defVal** is the default value of the transformed variable and if any of the input values are missing, **missingVal** is the value of the transformed variable.

The arguments InType, OutType, default_value and map_missing_to are optional. The CSV file containing the table should not have any row and column identifiers, and the values given must be in the same order as in the map command. If the data types of the variables are not given, the data types of the input variables are attempted to be determined from the **boxData** argument. If that is not possible, the data types are assumed to be string.

Intervals are either given by the left or right limits, in which case the other limit is considered as infinite. It may also be given by both the left and right limits separated by the character ":". An example of how intervals should be defined in the external file are:

```
rightVal1),outVal1
rightVal2],outVal2
[leftVal1:rightVal3),outVal3
(leftVal2:rightVal4],outVal4
(leftVal,outVal5
```

which, given an input value **inVal** and the output value to be calculated **out**, means that:

```
if(inVal < rightVal1) out=outVal1
f(inVal <= rightVal2) out=outVal2
if( (inVal >= leftVal1) and (inVal < rightVal3) ) out=outVal3
if( (inVal > leftVal2) and (inVal <= rightVal4) ) out=outVal4
if(inVal > leftVal) out=outVal5
```

It is also possible to give the information about the transforms without an external file, using a list of data frames. Each data frame defines a discretization operation for 1 input variable. The first row of the data frame gives the original field name, the derived field name, the left interval, the left value, the right interval and the right value. The second row gives the data type of the values as listed in the first row. The second row with the data types of the fields is not required. If not given, all fields are assumed to be strings. In this input format, the 'default_value' and 'map_missing_to' parameters should be vectors. The first element of each vector will correspond to the derived field defined in the 1st element of the 'xform_info' list etc. Although somewhat more complicated, this method is

designed to not require any external features. Further, once the initial list is constructed, modifying it is a simple operation; making this a better method to use if the parameters of the transformation are to be modified frequently and/or automatically. This is made more clear in the example below.

Value

R object containing the raw data, the transformed data and data statistics.

Author(s)

Tridivesh Jena

See Also

xform_wrap

Examples

```
# First wrap the data
iris_box <- xform_wrap(iris)
## Not run:
# Convert the continuous variable "Sepal.Length" to a discrete
# variable "dsl". The intervals to be used for this transformation is
# given in a file, "intervals.csv", whose content is, for example,:
#
#  5],val1
#  (5:6],22
#  (6,val2
#
# This will be used to create a discrete variable named "dsl" of dataType
# "string" such that:
#    if(Sepal.length <= 5) then dsl = "val1"
#    if((Sepal.Lenght > 5) and (Sepal.Length <= 6)) then dsl = "22"
#    if(Sepal.Length > 6) then dsl = "val2"
#
# Give "dsl" the value 0 if the input variable value is missing.
iris_box <- xform_discretize(iris_box,
  xform_info = "[Sepal.Length -> dsl][double -> string]",
  table = "intervals.csv", map_missing_to = "0"
)

## End(Not run)

# A different transformation using a list of data frames, of size 1:
t <- list()
m <- data.frame(rbind(
  c(
    "Petal.Length", "dis_pl", "leftInterval", "leftValue",
    "rightInterval", "rightValue"
  ),
  c(
    "double", "integer", "string", "double", "string",
```

```
    "double"
  ),
  c("0)", 0, "open", NA, "Open", 0),
  c(NA, 1, "closed", 0, "Open", 1),
  c(NA, 2, "closed", 1, "Open", 2),
  c(NA, 3, "closed", 2, "Open", 3),
  c(NA, 4, "closed", 3, "Open", 4),
  c("[4", 5, "closed", 4, "Open", NA)
), stringsAsFactors = TRUE)

# Give column names to make it look nice; not necessary!
colnames(m) <- c(
  "Petal.Length", "dis_pl", "leftInterval", "leftValue",
  "rightInterval", "rightValue"
)

# A textual representation of the data frame is:
#   Petal.Length  dis_pl leftInterval leftValue rightInterval rightValue
# 1 Petal.Length  dis_pl leftInterval leftValue rightInterval rightValue
# 2       double integer       string    double        string     double
# 3           0)       0         open      <NA>          Open          0
# 4         <NA>       1       closed         0          Open          1
# 5         <NA>       2       closed         1          Open          2
# 6         <NA>       3       closed         2          Open          3
# 7         <NA>       4       closed         3          Open          4
# 8           (4       5       closed         4          Open       <NA>
#
# This is a transformation that defines a derived field 'dis_pl'
# which has the integer value '0' if the original field
# 'Petal.Length' has a value less than 0. The derived field has a
# value '1' if the input is greater than or equal to 0 and less
# than 1. Note that the values of the 1st column after row 2 have
# been deliberately given NA values in the middle. This is to
# show that that column is meant for a textual representation of
# the transformation as defined for the method involving external
# files; however in this methodtheir values are not used.

# Add the data frame to a list. The default values and the missing
# values should be given as a vector, each element of the vector
# corresponding to the element at the same index in the list. If
# these values are not given as a vector, they will be used for the
# first list element only.
t[[1]] <- m
def <- c(11)
mis <- c(22)
iris_box <- xform_discretize(iris_box,
  xform_info = t, default_value = def,
  map_missing_to = mis
)

# Make a simple model to see the effect.
fit <- lm(Petal.Width ~ ., iris_box$data[, -5])
fit_pmml <- pmml(fit, transforms = iris_box)
```

`xform_function` *Add a function transformation to a xform_wrap object.*

Description

Add a function transformation to a xform_wrap object.

Usage

```
xform_function(
  wrap_object,
  orig_field_name,
  new_field_name = "newField",
  new_field_data_type = "numeric",
  expression,
  map_missing_to = NA
)
```

Arguments

`wrap_object`	Output of xform_wrap or another transformation function.
`orig_field_name`	String specifying name(s) of the original data field(s) being used in the transformation.
`new_field_name`	Name of the new field created by the transformation.
`new_field_data_type`	R data type of the new field created by the transformation ("numeric" or "factor").
`expression`	String expression specifying the transformation.
`map_missing_to`	Value to be given to the transformed variable if the value of any input variable is missing.

Details

Calculate the expression provided in `expression` for every row in the `wrap_object$data` data frame. The `expression` argument must represent a valid R expression, and any functions used in `expression` must be defined in the current environment.

The name of the new field is optional (a default name is provided), but an error will be thrown if attempting to create a field with a name that already exists in the xform_wrap object.

When `new_field_data_type = "numeric"`, the `DerivedField` attributes in PMML will be `dataType = "double"` and `optype = "continuous"`. When `new_field_data_type = "factor"`, these attributes will be `dataType = "string"` and `optype = "categorical"`.

Value

R object containing the raw data, the transformed data and data statistics. The `data` data frame will contain a new `new_field_name` column, and `field_data` will contain a new `new_field_name` row.

See Also

`xform_wrap`

Examples

```
# Load the standard iris dataset:
data(iris)

# Wrap the data:
iris_box <- xform_wrap(iris)

# Perform a transform on the Sepal.Length field:
# the value is squared and then divided by 100
iris_box <- xform_function(iris_box,
  orig_field_name = "Sepal.Length",
  new_field_name = "Sepal.Length.Transformed",
  expression = "(Sepal.Length^2)/100"
)

# Combine two fields to create another new feature:
iris_box <- xform_function(iris_box,
  orig_field_name = "Sepal.Width, Petal.Width",
  new_field_name = "Width.Sum",
  expression = "Sepal.Width + Sepal.Length"
)

# Create linear model using the derived features:
fit <- lm(Petal.Length ~
Sepal.Length.Transformed + Width.Sum, data = iris_box$data)

# Create pmml from the fit:
fit_pmml <- pmml(fit, transform = iris_box)
```

`xform_map` *Implement a map between discrete values in accordance with the PMML element* ***MapValues***.

Description

Implement a map between discrete values in accordance with the PMML element **MapValues**.

Usage

```
xform_map(
  wrap_object,
  xform_info,
  table = NA,
  default_value = NA,
  map_missing_to = NA,
  ...
)
```

Arguments

wrap_object	Output of xform_wrap or another transformation function.
xform_info	Specification of details of the transformation. It can be a text giving the external file name or a list of data frames. Even if only 1 variable is to be transformed, the information for that map should be given as a list with 1 element.
table	Name of external CSV file containing the map from input to output values.
default_value	The default value to be given to the transformed variable. If 'xform_info' is a list, this is a vector with each element corresponding to the corresponding list element.
map_missing_to	Value to be given to the transformed variable if the value of the input variable is missing. If 'xform_info' is a list, this is a vector with each element corresponding to the corresponding list element.
...	Further arguments passed to or from other methods.

Details

Map discrete values of an input variable to a discrete value of the transformed variable. The map can be given in an external table file referred to in the transform command or as a list of data frames, each data frame defining a map transform for one variable.

Given a map from the combination of variables **InVar1**, **InVar2**, ... to the transformed variable **OutVar**, where the variables have the data types **InType1**, **InType2**, ... and **OutType**, the map command is in the format:

```
xform_info = "[InVar1,InVar2,... -> OutVar][InType1,InType2,... -> OutType]"
table = "TableFileName", default_value = "defVal", map_missing_to = "missingVal"
```

where **TableFileName** is the name of the CSV file containing the map. The map can be a N to 1 map where N is greater or equal to 1. The data types of the variables can be any of the ones defined in the PMML format including integer, double or string. **defVal** is the default value of the transformed variable and if any of the map input values are missing, **missingVal** is the value of the transformed variable.

The arguments InType, OutType, default_value and map_missing_to are optional. The CSV file containing the table should not have any row and column identifiers, and the values given must be in the same order as in the map command. If the data types of the variables are not given, the data

types of the input variables are attempted to be determined from the **boxData** argument. If that is not possible, the data type is assumed to be string.

It is also possible to give the maps to be implemented without an external file using a list of data frames. Each data frame defines a map for 1 input variable. Given a data frame with N+1 columns, it is assumed that the map is a N to 1 map where the last column of the data frame corresponds to the derived field. The 1st row is assumed to be the names of the fields and the second row the data types of the fields. The rest of the rows define the map; each combination of the input values in a row is mapped to the value in the last column of that row. The second row with the data types of the fields is not required. If not given, all fields are assumed to be strings. In this input format, the 'default_value' and 'map_missing_to' parameters should be vectors. The first element of each vector will correspond to the derived field defined in the 1st element of the 'xform_info' list etc. These are made clearer in the example below.

Value

R object containing the raw data, the transformed data and data statistics.

Author(s)

Tridivesh Jena

See Also

xform_wrap, pmml

Examples

```
# Load the standard audit dataset, part of the pmml package:
data(audit)

# First wrap the data:
audit_box <- xform_wrap(audit)
## Not run:
# One of the variables, "Sex", has 2 possible values: "Male"
# and "Female". If these string values have to be mapped to a
# numeric value, a file has to be created, say "map_audit.csv",
# whose content is, for example:
#
#  Male,1
#  Female,2
#
# Transform the variable "Gender" to a variable "d_gender"
# such that:
#    if Sex = "Male" then d_sex = "1"
#    if Sex = "Female" then d_sex = "2"
#
# Give "d_sex" the value 0 if the input variable value is
# missing.
audit_box <- xform_map(audit_box,
  xform_info = "[Sex -> d_sex][string->integer]",
  table = "map_audit.csv", map_missing_to = "0"
```

```
)

## End(Not run)
# Same as above, with an extra variable, but using data frames.
# The top 2 rows give the variable names and their data types.
# The rest represent the map. For example, the third row
# indicates that when the input variable "Sex" has the value
# "Male" and the input variable "Employment" has
# the value "PSLocal", the output variable "d_sex" should have
# the value 1.
t <- list()
m <- data.frame(
  c("Sex", "string", "Male", "Female"),
  c("Employment", "string", "PSLocal", "PSState"),
  c("d_sex", "integer", 1, 0),
  stringsAsFactors = TRUE
)
t[[1]] <- m

# Give default value as a vector and missing value as a string,
# this is only possible as there is only one map defined. If
# default values is not given, it will simply not be given in
# the PMML file as well. In general, the default values and the
# missing values should be given as a vector, each element of
# the vector corresponding to the element at the same index in
# the list. If these values are not given as a vector, they will
# be used for the first list element only.
audit_box <- xform_map(audit_box,
  xform_info = t, default_value = c(3),
  map_missing_to = "2"
)

# check what the pmml looks like
fit <- lm(Adjusted ~ ., data = audit_box$data)
fit_pmml <- pmml(fit, transforms = audit_box)
```

`xform_min_max` *Normalize continuous values in accordance with the PMML element* ***NormContinuous***.

Description

Normalize continuous values in accordance with the PMML element **NormContinuous**.

Usage

```
xform_min_max(wrap_object, xform_info = NA, map_missing_to = NA, ...)
```

Arguments

wrap_object	Output of xform_wrap or another transformation function.
xform_info	Specification of details of the transformation.
map_missing_to	Value to be given to the transformed variable if the value of the input variable is missing.
...	Further arguments passed to or from other methods.

Details

Given input data in a xform_wrap format, normalize the given data values to lie between provided limits.

Given an input variable named **InputVar**, the name of the transformed variable **OutputVar**, the desired minimum value the transformed variable may have **low_limit**, the desired maximum value the transformed variable may have **high_limit**, and the desired value of the transformed variable if the input variable value is missing **missingVal**, the **xform_min_max** command including all the optional parameters is in the format:

```
formInfo = "InputVar -> OutputVar[low_limit,high_limit]"
map_missing_to = "missingVal"
```

There are two ways to refer to variables. The first way is to use the variable's column number; given the **data** attribute of the **boxData** object, this would be the order at which the variable appears. This can be indicated in the format "column#". The second way is to refer to the variable by its name.

The name of the transformed variable is optional; if the name is not provided, the transformed variable is given the name: "derived_" + *original_variable_name*. Similarly, the low and high limit values are optional; they have the default values of 0 and 1 respectively. **missingValue** is an optional parameter as well. It is the value of the derived variable if the input value is missing.

If no input variable names are provided, by default all numeric variables are transformed. Note that in this case a replacement value for missing input values cannot be specified; the same applies to the **low_limit** and **high_limit** parameters.

Value

R object containing the raw data, the transformed data and data statistics.

Author(s)

Tridivesh Jena

See Also

xform_wrap

Examples

```
# Load the standard iris dataset:
data(iris)

# First wrap the data:
iris_box <- xform_wrap(iris)

# Normalize all numeric variables of the loaded iris dataset to lie
# between 0 and 1. These would normalize "Sepal.Length", "Sepal.Width",
# "Petal.Length", "Petal.Width" to the 4 new derived variables named
# derived_Sepal.Length, derived_Sepal.Width, derived_Petal.Length,
# derived_Petal.Width.
iris_box_1 <- xform_min_max(iris_box)

# Normalize the 1st column values of the dataset (Sepal.Length) to lie
# between 0 and 1 and give the derived variable the name "dsl".
iris_box_1 <- xform_min_max(iris_box, xform_info = "column1 -> dsl")

# Repeat the above operation; adding the new transformed variable to
# the iris_box object.
iris_box <- xform_min_max(iris_box, xform_info = "column1 -> dsl")

# Transform Sepal.Width(the 2nd column).
# The new transformed variable will be given the default name
# "derived_Sepal.Width".
iris_box_3 <- xform_min_max(iris_box, xform_info = "column2")

# Repeat the same operation as above, this time using the variable name.
iris_box_4 <- xform_min_max(iris_box, xform_info = "Sepal.Width")

# Repeat the same operation as above, now assigning the transformed variable,
# "derived_Sepal.Width", the value of 0.5 if the input value of the
# "Sepal.Width" variable is missing.
iris_box_5 <- xform_min_max(iris_box, xform_info = "Sepal.Width", "map_missing_to=0.5")

# Transform Sepal.Width(the 2nd column) to lie between 2 and 3.
# The new transformed variable will be given the default name
# "derived_Sepal.Width".
iris_box_6 <- xform_min_max(iris_box, xform_info = "column2->[2,3]")

# Repeat the above transformation, this time the transformed variable
# lies between 0 and 10.
iris_box_7 <- xform_min_max(iris_box, xform_info = "column2->[,10]")
```

`xform_norm_discrete` *Normalize discrete values in accordance with the PMML element* ***NormDiscrete****.*

Description

Normalize discrete values in accordance with the PMML element **NormDiscrete**.

Usage

```
xform_norm_discrete(
  wrap_object,
  xform_info = NA,
  input_var = NA,
  map_missing_to = NA,
  ...
)
```

Arguments

wrap_object	Output of xform_wrap or another transformation function.
xform_info	Specification of details of the transformation: the name of the input variable to be transformed.
input_var	The input variable name in the data on which the transformation is to be applied.
map_missing_to	Value to be given to the transformed variable if the value of the input variable is missing.
...	Further arguments passed to or from other methods.

Details

Define a new derived variable for each possible value of a categorical variable. Given a categorical variable **catVar** with possible discrete values **A** and **B**, this will create 2 derived variables **catVar_A** and **catVar_B**. If, for example, the input value of **catVar** is **A** then **catVar_A** equals 1 and **catVar_B** equals 0.

Given an input variable, **input_var** and **missingVal**, the desired value of the transformed variable if the input variable value is missing, the xform_norm_discrete command including all optional parameters is in the format:

xform_info="input_var=input_variable, map_missing_to=missingVal"

There are two methods in which the input variable can be referred to. The first method is to use its column number; given the **data** attribute of the **boxData** object, this would be the order at which the variable appears. This can be indicated in the format "column#". The second method is to refer to the variable by its name.

The **xform_info** and **input_var** parameters provide the same information. While either one may be used when using this function, at least one of them is required. If both parameters are given, the **input_var** parameter is used as the default.

The output of this transformation is a set of transformed variables, one for each possible value of the input variable. For example, given possible values of the input variable **val1**, **val2**, ... these transformed variables are by default named **input_var_val1**, **input_var_val2**, ...

Value

R object containing the raw data, the transformed data and data statistics.

Author(s)

Tridivesh Jena

See Also

xform_wrap

Examples

```
# Load the standard iris dataset, already available in R
data(iris)

# First wrap the data
iris_box <- xform_wrap(iris)

# Discretize the "Species" variable. This will find all possible
# values of the "Species" variable and define new variables. The
# parameter name used here should be replaced by the new preferred
# parameter name as shown in the next example below.
#
#  "Species_setosa" such that it is 1 if
#      "Species" equals "setosa", else 0;
#  "Species_versicolor" such that it is 1 if
#      "Species" equals "versicolor", else 0;
#  "Species_virginica" such that it is 1 if
#      "Species" equals "virginica", else 0

iris_box <- xform_norm_discrete(iris_box, input_var = "Species")

# Exact same operation performed with a different parameter name.
# Use of this new parameter is the preferred method as the previous
# parameter will be deprecated soon.

iris_box <- xform_wrap(iris)
iris_box <- xform_norm_discrete(iris_box, xform_info = "Species")
```

---

xform_wrap *Wrap data in a data transformations object.*

Description

Wrap data in a data transformations object.

Usage

```
xform_wrap(data, use_matrix = FALSE)
```

Arguments

data	The raw data set.
use_matrix	Boolean value indicating whether data should be stored in matrix format as well.

Details

Wrap raw data read in an R object. This object can then be passed to various transform functions, and the data in it transformed.

The object consists of the data itself and various properties for each data variable. Since the data is not always required to be in matrix format as well as a data frame, the 'use_matrix' value lets the user decide if the data should be stored in both formats, giving the user a choice in reducing the speed of the transformation operations and the memory required. If there is not enough information about the data, they are given default values; the data is assumed to be the original data of data type string. The variable names are assumed to be **X1**, **X2**, ... This information is then used by the transformation functions to calculate the derived variable values.

Value

An R object containing information on the data to be transformed.

Author(s)

Tridivesh Jena

See Also

pmml

Examples

```
# Load the standard iris dataset
data(iris)

# Make a object for the iris dataset to use with
# transformation functions
iris_box <- xform_wrap(iris)

# Output only the transformations in PMML format.
# This example will output just an empty "LocalTransformations"
# element as no transformations were performed.
trans_pmml <- pmml(NULL, transforms = iris_box)

# The following will also work
trans_pmml_2 <- pmml(, transforms = iris_box)
```

---

xform_z_score *Perform a z-score normalization on continuous values in accordance with the PMML element **NormContinuous**.*

Description

Perform a z-score normalization on continuous values in accordance with the PMML element **NormContinuous**.

Usage

```
xform_z_score(wrap_object, xform_info = NA, map_missing_to = NA, ...)
```

Arguments

wrap_object	Output of xform_wrap or another transformation function.
xform_info	Specification of details of the transformation.
map_missing_to	Value to be given to the transformed variable if the value of the input variable is missing.
...	Further arguments passed to or from other methods.

Details

Perform a z-score normalization on data given in `xform_wrap` format.

Given an input variable named **InputVar**, the name of the transformed variable **OutputVar**, and the desired value of the transformed variable if the input variable value is missing **missingVal**, the xform_z_score command including all the optional parameters is:

`xform_info="InputVar -> OutputVar", map_missing_to="missingVal"`

Two methods can be used to refer to the variables. The first method is to use its column number; given the **data** attribute of the **boxData** object, this would be the order at which the variable appears. This can be indicated in the format "column#". The second method is to refer to the variable by its name.

The name of the transformed variable is optional; if the name is not provided, the transformed variable is given the name: "derived_" + *original_variable_name*

missingValue, an optional parameter, is the value to be given to the output variable if the input variable value is missing. If no input variable names are provided, by default all numeric variables are transformed. Note that in this case a replacement value for missing input values cannot be specified.

Value

R object containing the raw data, the transformed data and data statistics.

Author(s)

Tridivesh Jena

See Also

`xform_wrap`

Examples

```
# Load the standard iris dataset
data(iris)

# First wrap the data
```

```
iris_box <- xform_wrap(iris)

# Perform a z-transform on all numeric variables of the loaded
# iris dataset. These would be Sepal.Length, Sepal.Width,
# Petal.Length, and Petal.Width. The 4 new derived variables
# will be named derived_Sepal.Length, derived_Sepal.Width,
# derived_Petal.Length, and derived_Petal.Width
iris_box_1 <- xform_z_score(iris_box)

# Perform a z-transform on the 1st column of the dataset (Sepal.Length)
# and give the derived variable the name "dsl"
iris_box_2 <- xform_z_score(iris_box, xform_info = "column1 -> dsl")

# Repeat the above operation; adding the new transformed variable
# to the iris_box object
iris_box <- xform_z_score(iris_box, xform_info = "column1 -> dsl")

# Transform Sepal.Width(the 2nd column)
# The new transformed variable will be given the default name
# "derived_Sepal.Width"
iris_box_3 <- xform_z_score(iris_box, xform_info = "column2")

# Repeat the same operation as above, this time using the variable
# name
iris_box_4 <- xform_z_score(iris_box, xform_info = "Sepal.Width")

# Repeat the same operation as above, assign the transformed variable
# "derived_Sepal.Width". The value of 1.0 if the input value of the
# "Sepal.Width" variable is missing. Add the new information to the
# iris_box object.
iris_box <- xform_z_score(iris_box,
  xform_info = "Sepal.Width",
  "map_missing_to=1.0"
)
```

Index