Package 'pmml'

January 25, 2019

Type Package

Title Generate PMML for Various Models

Version 1.5.7

Author Graham Williams, Tridivesh Jena, Wen Ching Lin, Michael Hahsler (arules), Software AG, Hemant Ishwaran, Udaya B. Kogalur, Rajarshi Guha, Dmitriy Bolotov

Maintainer Tridivesh Jena <rpmmlsupport@softwareag.com>

Depends XML

Suggests ada, amap, arules, gbm, glmnet, neighbr, nnet, rpart, randomForestSRC (<= 2.5.0), randomForest, kernlab, e1071, testthat, survival, xgboost, pmmlTransformations(>= 1.3.1), knitr, rmarkdown

Imports methods, stats, utils, stringr

License GPL (>= 2.1)

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://www.dmg.org>. The generated PMML can be imported into any PMML consuming application, such as Zementis Predictive Analytics products, which integrate with web services, relational database systems and deploy natively on Hadoop in conjunction with Hive, Spark or Storm, as well as allow predictive analytics to be executed for IBM z Systems mainframe applications and real-time, streaming analytics platforms. The package isofor (used for anomaly detection) can be installed with devtools::install_github(``Zelazny7/isofor'').

URL https://www.softwareag.com/zementis

NeedsCompilation no

RoxygenNote 6.1.1

VignetteBuilder knitr

Repository CRAN

Date/Publication 2019-01-25 07:30:03 UTC

R topics documented:

AddAttributes *adds attribute values to an existing element in a given PMML file*

Description

This helper function allows one to add attributes to an arbitrary xml element. This is an experimental function designed to be more general than the 'addMSAttributes' or 'addDDAttributes' functions.

Usage

```
AddAttributes(xmlmodel=NULL, xpath=NULL, attributes=NULL,
              namespace="4_3",...)
```

Arguments

xmlmodel	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 fileToXMLNode 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 the model is represented as.
...	further arguments passed to or from other methods.

Details

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@softwareag.com>

Examples

```
# make a sample model
library(pmml)
model0 <- lm(Sepal.Length~., data=iris[,-5])
model <- pmml(model0)

# The resulting PMML:
# <PMML version="4.3" ... xmlns="http://www.dmg.org/PMML-4_3">
# <Header ... description="Linear Regression Model"/>
# <DataDictionary numberOfFields="4">
# .
# .
# </DataDictionary>
# <RegressionModel modelName="Linear_Regression_Model"
#     functionName="regression"
#     algorithmName="least squares">
#  <MiningSchema>
#  .
#  .
#  </MiningSchema>
# .
# .
#  <RegressionTable intercept="1.85599749291755">
#   <NumericPredictor name="Sepal.Width" exponent="1"
#    coefficient="0.650837159313218"/>
#   <NumericPredictor name="Petal.Length" exponent="1"
#    coefficient="0.709131959136729"/>
#   <NumericPredictor name="Petal.Width" exponent="1"
#    coefficient="-0.556482660167024"/>
#  </RegressionTable>
# </RegressionModel>
# </PMML>

# 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.
AddAttributes(model, "/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.
AddAttributes(model,
   "/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.
AddAttributes(model,
   "/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'.
AddAttributes(model,
   "/p:PMML/descendant::p:NumericPredictor[contains(@name,'Length')]",
   attributes=c(a=1,b="b"))
```

---

addDDAttributes *adds attribute values to an existing DataField element in a given PMML file*

Description

The PMML format allows a DataField element to have various attributes 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 DataFields.

Usage

```
addDDAttributes(xmlmodel=NULL,attributes=NULL,field=NULL,
                namespace="4_3",...)
```

Arguments

xmlmodel	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 fileToXMLNode 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 the model is represented as.
...	further arguments passed to or from other methods.

Details

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 convinient 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@softwareag.com>

Examples

```
# make a sample model
library(pmml)
model0 <- lm(Sepal.Length~., data=iris[,-5])
model <- pmml(model0)

# 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),
                         stringAsFactors=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)

# actual command
addDDAttributes(model,attributes,namespace="4_3")

# Alternative method to add attributes to a single field,
# "Sepal.Width"
addDDAttributes(model,c(displayName="FlowerWidth",isCyclic=1),
               "Sepal.Width")


mi<-makeIntervals(list("openClosed","closedClosed","closedOpen"),
                  list(NULL,1,2),list(1,2,NULL))
mv<-makeValues(list("A","B","C"),list(NULL,NULL,NULL),
                  list("valid",NULL,"invalid"))
addDFChildren(model, field="Sepal.Length", interval=mi, values=mv)
```

addDFChildren *adds 'Interval' and 'Value' child elements to a given DataField element in a given PMML file*

Description

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.

Usage

```
addDFChildren(xmlmodel=NULL,field=NULL,intervals=NULL,
              values=NULL,namespace="4_3",...)
```

Arguments

xmlmodel	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 fileToXMLNode 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 the model is represented as.
...	further arguments passed to or from other methods.

Details

The 'Interval' elements or the 'Value' elements can be typed in, but more conviniently created by using the helper functions 'makeIntervals' 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@softwareag.com>

Examples

```
# make a sample model
library(pmml)
model0 <- lm(Sepal.Length~., data=iris[,-5])
model <- pmml(model0)

# 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
addDFChildren(model, 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<-makeIntervals(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<-makeValues(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.
addDFChildren(model, field="Sepal.Length", intervals=mi, values=mv)
# Only defined 'Interval's
addDFChildren(model, field="Sepal.Length", intervals=mi)
```

---

addLT *adds a LocalTransformations element to a given PMML file.*

Description

The pmmlTransformations package allows one to create a LocalTransformations element describing the data manipulations desired. This function allows one to add this information to a given PMML file; thus combining the description of any data processing as well as the model using such transformed data.

Usage

```
addLT(xmlmodel=NULL, transforms=NULL, namespace="4_3",...)
```

Arguments

xmlmodel	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 fileToXMLNode function.
transforms	the transformations performed on the initial data. This is the LocalTransformations element as an XML node object.
namespace	the namespace of the PMML model. This is frequently also the PMML version the model is represented as.
...	further arguments passed to or from other methods.

Details

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 fifferent 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@softwareag.com>

Examples

```
## Not run:
# make a sample model

library(pmml)
model <- pmml(lm(Sepal.Length~., data=iris[,-5]))

# Perform a z-score transform on the first variable of the data set.
# As it is created and used in the same R session, 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.

library(pmmlTransformations)
irisBox <- WrapData(iris)
irisBox <- ZScoreXform(irisBox,"1")
xforms <- pmml(,transforms=irisBox)

# Add the LocalTransformations element to the initial PMML model.
```

```
# Since the model still uses the original fields, the usage
# envisioned for this function is to make it easy if the modeller
# forgot to add the transformations when using the pmml function
# initially.

modified <- addLT(model,xforms,namespace="4_3")

## End(Not run)
```

`addMSAttributes` *adds attribute values to an existing MiningField element in a given PMML file*

Description

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.

Usage

```
addMSAttributes(xmlmodel=NULL,attributes=NULL,
                namespace="4_3",...)
```

Arguments

`xmlmodel`	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 fileToXMLNode 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 the model is represented as.
`...`	further arguments passed to or from other methods.

Details

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@softwareag.com>

Examples

```
# make a sample model
library(pmml)
model0 <- lm(Sepal.Length~., data=iris[,-5])
model <- pmml(model0)

# 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"))
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]])
 }

# actual command
addMSAttributes(model,attributes,namespace="4_3")
```

---

addOutputField *Add Output nodes to a PMML object.*

Description

Add Output nodes to a PMML object.

Usage

```
addOutputField(xmlmodel = NULL, outputNodes = NULL, at = "End",
  xformText = NULL, nodeName = NULL, attributes = NULL,
  whichOutput = 1, namespace = "4_3")
```

Arguments

xmlmodel	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 'makeOutputNodes' 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 functionToPMML 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 'makeOutputNodes' 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 'xmlmodel', '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 functionToPMML function. In other words, find the OutputField node with tha 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<-makeOutputNodes(name=list("OutputField","OutputField"),
                         attributes=list(list(name="dbl",
                         optype="continuous"),NULL),
                         expression=list("ln(x)","ln(x/(1-x))"))
onodes2<-makeOutputNodes(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
addOutputField(xmlmodel=pmod, outputNodes=onodes2, at="End",
               xformText=NULL, nodeName=NULL, attributes=NULL,
               whichOutput=1)
pmod2<-addOutputField(xmlmodel=pmod, outputNodes=onodes0, at="End",
                      xformText=NULL, nodeName=NULL,
                      attributes=NULL,whichOutput=1)

# Create nodes with attributes and transformations
addOutputField(xmlmodel=pmod2, outputNodes=onodes2,at=2)
addOutputField(xmlmodel=pmod2, xformText=list("exp(x) && !x"),
               nodeName="Predicted_Sepal.Length")

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

---

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***
http://www.dmg.org/pmml_examples/index.html#Audit

Examples

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

`fileToXMLNode` *Reads in a file and tries to parse it into an object of type XMLNode*

Description

This function can be used when the user wants to read in an external file and convert it into an XMLNode to be used subsequently by other R functions.

Usage

```
fileToXMLNode(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

This function reads in a file and attempts to parse it into an XML node. 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 'fileToXMLNode' 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@softwareag.com>

Examples

```
## Not run:
# define some transformations
 library(pmml)
 library(pmmlTransformations)

 irisBox <- WrapData(iris)
 irisBox <- ZScoreXform(irisBox,xformInfo = "column1->d1")
 irisBox <- ZScoreXform(irisBox,xformInfo = "column2->d2")

#make a LocalTransformations element and save it to an external file
 pmml_trans <- pmml(NULL, transforms=irisBox)
 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 <- fileToXMLNode("xform_iris.pmml")

## End(Not run)
```

---

functionToPMML *Convert an R expression to PMML.*

Description

Convert an R expression to PMML.

Usage

```
functionToPMML(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 ‘FunctionXform‘ 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 ‘functionToPMML()‘ 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
functionToPMML("1 + 3/5 - (4 * 2)")

# Nested arbitrary functions
functionToPMML("foo(bar(x)) - bar(foo(y-z))")

# If-else expression
functionToPMML("if (x==3) { 3 } else { 0 }")

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

# Function with string argument types
functionToPMML("colors('red','green','blue')")

# Sign in front of expression
functionToPMML("-(x/y)")
```

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")
```

`makeIntervals` *Create Interval elements, most likely to add to a DataDictionary element*

Description

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

Usage

```
makeIntervals(closure = NULL, leftMargin = NULL, rightMargin = NULL,
  namespace = "4_3")
```

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 'makeIntervals' 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

makeValues to make Values child elements, addDFChildren 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<-makeIntervals(list("openClosed","openOpen","closedOpen"),
                  list(NULL,1,2),list(1,2,NULL))
```

---

makeOutputNodes *Add Output nodes to a PMML object.*

Description

Add Output nodes to a PMML object.

Usage

```
makeOutputNodes(name = "OutputField", attributes = NULL,
  expression = NULL, namespace = "4_3")
```

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 the functionToPMML function
namespace	The namespace of the PMML model

Details

This function will create a list of nodes with names 'name', attributes 'attributes' and child elements 'expression'. 'expression' is a string converted to XML similar to th functionToPMML function. Meant to create OutputField elements, 'expressions' allows one to include 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 2 nodes, one with attributes
TwoNodes <- makeOutputNodes(name=list("OutputField","OutputField"),
             attributes=list(list(name="dbl",optype="continuous"),NULL),
             expression=list("ln(x)","ln(x/(1-x))"))
```

---

makeValues *Create Values element, most likely to add to a DataDictionary element*

Description

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

Usage

```
makeValues(value = NULL, displayValue = NULL, property = NULL,
  namespace = "4_3")
```

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

The 'makeValues' function is used the same way as the 'makeIntervals' 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

makeIntervals to make Interval child elements, addDFChildren 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 <- makeValues(list(1.1,2.2,3.3),list(NULL,NULL,NULL),
                 list("valid",NULL,"invalid"))
```

---

pmml *Generate PMML 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.

The same 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 pmmlTransformations object as the transforms input. Please see the R **pmmlTransformations** package for more information on how to create the pmmlTransformations object.

In addition, the pmml function can also be called using a pre-existing PMML model as the first input and a pmmlTransformations 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 mutiple fields derived from the same original field.

This package converts models to PMML version 4.3.

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

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 packages is available in the vignette:

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

Usage

```
pmml(model=NULL, model.name="Rattle_Model",
     app.name="Rattle/PMML", description=NULL,
     copyright=NULL, transforms=NULL, ...)
```

Arguments

model	an object to be converted to PMML.
model.name	a name to be given to the model in the PMML code.
app.name	the name of the application that generated the PMML code.
description	a descriptive text for the Header element of the PMML code.
copyright	the copyright notice for the model.
transforms	data transformations represented in PMML via **pmmlTransformations**.
...	further arguments passed to or from other methods.

Details

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://www.dmg.org.

The generated PMML can be imported into any PMML consuming application, such as Zementis Predictive Analytics products, which integrate with web services, relational database systems and deploy natively on Hadoop in conjunction with Hive, Spark or Storm, as well as allow predictive analytics to be executed for IBM z Systems mainframe applications and real-time, streaming analytics platforms.

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@togaware.com>

References

- Rattle home page: https://rattle.togaware.com/
- PMML home page: http://www.dmg.org
- A. Guazzelli, W. Lin, T. Jena (2012), *PMML in Action: Unleashing the Power of Open Standards for Data Mining and Predictive Analytics*. CreativeSpace (Second Edition) - Available on Amazon.com.
- A. Guazzelli, M. Zeller, W. Lin, G. Williams (2009), PMML: An Open Standard for Sharing Models. *The R journal*, Volume 1/1, 60-65
- 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*
- T. Jena, A. Guazzelli, W. Lin, M. Zeller (2013). The R pmmlTransformations Package. In *Proceedings of the 19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*

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.randomForest, pmml.rfsrc, pmml.rpart, pmml.svm, pmml.xgb.Booster

Examples

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

# Convert to pmml
pmml(iris.lm)

# Create a pmmlTransformations object
library(pmmlTransformations)
xo <- WrapData(iris)

# Transform the 'Sepal.Length' variable
xo <- MinMaxXform(xo,xformInfo="column1->d_sl")

# Output the tranformation in PMML format
pmml(NULL, transforms=xo)
```

pmml.ada *Generate PMML for ada objects*

Description

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

Usage

```
## S3 method for class 'ada'
pmml(model, model.name="AdaBoost_Model",
    app.name="R-PMML", description="AdaBoost Model",
    copyright=NULL, transforms=NULL, unknownValue=NULL, ...)
```

Arguments

model	ada object.
model.name	a name to be given to the model in the PMML code.
app.name	the name of the application that generated the PMML code.
description	a descriptive text for the Header element of the PMML code.
copyright	the copyright notice for the model.
transforms	data transformations represented in PMML via **pmmlTransformations**.
unknownValue	value to be used as the 'missingValueReplacement' attribute for all MiningFields.
...	further arguments passed to or from other methods.

Details

The pmml function exports 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)

Zementis Inc.

References

R project CRAN package: ***ada****: an R package for stochastic boosting*
https://CRAN.R-project.org/package=ada

Examples

```
library(ada)
library(pmml)
data(audit)

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

---

pmml.coxph *Generate PMML for coxph objects*

Description

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

Usage

```
## S3 method for class 'coxph'
pmml(model, model.name="CoxPH_Survival_Regression_Model",
    app.name="Rattle/PMML",
    description="CoxPH Survival Regression Model",
    copyright=NULL, transforms=NULL, unknownValue=NULL, ...)
```

Arguments

model	a coxph object.
model.name	a name to be given to the model in the PMML code.
app.name	the name of the application that generated the PMML code.
description	a descriptive text for the Header element of the PMML code.
copyright	the copyright notice for the model.
transforms	data transformations represented in PMML via **pmmlTransformations**.
unknownValue	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@togaware.com>, Zementis Inc.

References

R project CRAN package: ***survival**: Survival Analysis*
https://CRAN.R-project.org/package=survival

---

pmml.cv.glmnet *Generate PMML for glmnet objects*

Description

Generate the PMML representation for a glmnet (elasticnet general linear regression) object. In particular, this gives the PMML representation for an object created by the cv.glmnet function.

Usage

```
## S3 method for class 'cv.glmnet'
pmml(model, model.name="Elasticnet_Model",
    app.name="Rattle/PMML",
    description="Generalized Linear Regression Model",
    copyright=NULL, transforms=NULL, unknownValue=NULL,
    dataset=NULL, s=NULL, ...)
```

Arguments

model	a cv.glmnet object contained in an object of class **glmnet**, as contained in the object returned by the function cv.glmnet.
model.name	a name to be given to the model in the PMML code.
app.name	the name of the application that generated the PMML code.
description	a descriptive text for the Header element of the PMML code.
copyright	the copyright notice for the model.
transforms	data transformations represented in PMML via **pmmlTransformations**.
unknownValue	value to be used as the 'missingValueReplacement' attribute for all MiningFields.
dataset	the dataset using which the model was built.

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, such as from the **pmmlTransformations** 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", ...

Use of PMML and pmml.cv.glmnet requires the **XML** package. Be aware that XML is a very verbose data format.

Currently, only gaussian and poisson family types are supported.

Author(s)

Zementis Inc.

References

R project CRAN package:
***glmnet**: Lasso and elastic-net regularized generalized linear models*
https://CRAN.R-project.org/package=glmnet

Examples

```
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
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 in PMML format the regression model at the lambda
# parameter = 0.006
pmml(model2,s=0.006)
```

---

pmml.gbm *Generate PMML for genralized boosting tree objects*

Description

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

Usage

```
## S3 method for class 'gbm'
pmml(model, model.name="gbm_Model", app.name="R/PMML",
    description="Generalized Boosted Tree Model", copyright=NULL, transforms=NULL,
     unknownValue=NULL,  ...)
```

Arguments

model	a gbm object.
model.name	a name to be given to the model in the PMML code.
app.name	the name of the application that generated the PMML code.
description	a descriptive text for the Header element of the PMML code.
copyright	the copyright notice for the model.
transforms	data transformations represented in PMML via package **pmmlTransformations**.
unknownValue	value to be used as the 'missingValueReplacement' attribute for all MiningFields.
...	further arguments passed to or from other methods.

Details

This is an optimized function which outputs a gbm object produced by the **gbm** package in PMML format. This output is processed to minimize the time and memory requirements.The model will include not just the model but also any pre-processing applied to the training data.

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".

Author(s)

<tridivesh.jena@softwareag.com>

References

R project CRAN package:
***gbm**: Generalized Boosted Regression Models*
https://CRAN.R-project.org/package=gbm

Examples

```
# Build a simple gbm model

library(gbm)
library(pmml)
data(audit)

mod<-gbm(Adjusted~.,data=audit[,-c(1,4,6,9,10,11,12)],n.trees=3,interaction.depth=4)
# since distribution type is not given, a bernoulli distribution will be assumed

# Convert to pmml
pmml(mod)

# now try a classification case
mod2<-gbm(Species~.,data=iris,n.trees=2,interaction.depth=3,distribution="multinomial")

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

---

pmml.glm *Generate PMML for glm objects*

Description

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

Usage

```
## S3 method for class 'glm'
pmml(model, model.name="General_Regression_Model",
      app.name="Rattle/PMML",
      description="Generalized Linear Regression Model",
      copyright=NULL, transforms=NULL, unknownValue=NULL,
      weights=NULL, ...)
```

Arguments

model	a glm object.
model.name	a name to be given to the model in the PMML code.

app.name	the name of the application that generated the PMML code.
description	a descriptive text for the Header element of the PMML code.
copyright	the copyright notice for the model.
transforms	data transformations represented in PMML via **pmmlTransformations**.
unknownValue	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.

Author(s)

Zementis Inc.

References

R project: ***Fitting Generalized Linear Models***

Examples

```
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)

##  creating 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)
```

`pmml.hclust` *Generate PMML for hclust objects*

Description

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

Usage

```
## S3 method for class 'hclust'
pmml(model, model.name="HClust_Model", app.name="Rattle/PMML",
     description="Hierarchical cluster model", copyright=NULL,
     transforms=NULL, unknownValue=NULL, centers, ...)
```

Arguments

`model`	a hclust object.
`model.name`	a name to be given to the model in the PMML code.
`app.name`	the name of the application that generated the PMML code.
`description`	a descriptive text for the Header element of the PMML code.
`copyright`	the copyright notice for the model.
`transforms`	data transformations represented in PMML via **pmmlTransformations**.
`unknownValue`	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 repectively.

Author(s)

<Graham.Williams@togaware.com>

References

R project: ***Hierarchical Clustering***

Examples

```
## Not run:
# cluster the 4 numeric variables of the iris dataset
library(amap)
model <- hclusterpar(iris[,-5])

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

# convert to pmml
library(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 = "R", description = "Isolation Forest", copyright = NULL,
  transforms = NULL, unknownValue = NULL, anomalyThreshold = 0.6,
  parentInvalidValueTreatment = "returnInvalid",
  childInvalidValueTreatment = "asIs", ...)
```

Arguments

`model`	an iForest object from package **isofor**
`model.name`	optional; the model name.
`app.name`	optional; name where the model was created.
`description`	optional; description of the model.
`copyright`	optional; a copyright statement.

`transforms` optional; any pre-processing information from the pmmlTransformations package.

`unknownValue` optional; a missing value replacement.

`anomalyThreshold` double between 0 and 1. Predicted values greater than this are classified as anomalies.

`parentInvalidValueTreatment` invalid value treatment at the top MiningField level.

`childInvalidValueTreatment` invalid value treatment at the model segment MiningField level.

`...` further arguments passed to 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 `anomalyThreshold`, 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.

See Also

`pmml`, isofor package

Examples

```
## Not run:
# Any anomalous data points in the Iris dataset? Hopefully none of the
# anomaly scores are high!
library(isofor)
# create an isolation forest with 10 trees. Sample 30 data points at a time
# from the iris dataset to fit the trees
mod <- iForest(iris,nt=10,phi=30)
#convert to PMML
pm <- pmml(mod)

## End(Not run)
```

pmml.kmeans *Generate PMML for kmeans objects*

Description

Generate the PMML representation for a kmeans object (cluster) from package **stats**. 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="Rattle/PMML",
     description="KMeans cluster model", copyright=NULL,
     transforms=NULL, unknownValue=NULL,
     algorithm.name="KMeans: Hartigan and Wong", ...)
```

Arguments

model	a kmeans object.
model.name	a name to be given to the model in the PMML code.
app.name	the name of the application that generated the PMML code.
description	a descriptive text for the Header element of the PMML code.
copyright	the copyright notice for the model.
transforms	data transformations represented in PMML via **pmmlTransformations**.
unknownValue	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 the pmmlTransformations package so that the normalization information is included in the pmml model format.

Author(s)

<Graham.Williams@togaware.com>

References

R project: ***K-Means Clustering***

Examples

```
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)
pmml(cl)
```

`pmml.ksvm` *Generate PMML for ksvm objects*

Description

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

Usage

```
## S3 method for class 'ksvm'
pmml(model, model.name="SVM_model",
      app.name="Rattle/PMML",
      description="Support Vector Machine PMML Model",
      copyright=NULL, transforms=NULL, unknownValue=NULL,
      dataset=NULL, ...)
```

Arguments

`model`	a ksvm object.
`model.name`	a name to be given to the model in the PMML code.
`app.name`	the name of the application that generated the PMML code.
`description`	a descriptive text for the Header element of the PMML code.
`copyright`	the copyright notice for the model.
`transforms`	data transformations represented in PMML via **pmmlTransformations**.
`unknownValue`	value to be used as the 'missingValueReplacement' attribute for all MiningFields.
`dataset`	required since the ksvm object does not record information about the used categorical variable; the original dataset used to train the SVM model in ksvm.
`...`	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.

Author(s)

Zementis Inc.

References

R project CRAN package: ***kernlab****: Kernel-based Machine Learning Lab*
https://CRAN.R-project.org/package=kernlab

Examples

```
# Train a support vector machine to perform classification.
library(kernlab)
model  <- ksvm(Species ~ ., data=iris)
p <- pmml(model, dataset=iris)

# To make predictions using this model, the new data must be given;
# without it and by simply using the "predict" function without an
# input dataset, the predicted value will not be the true predicted
# value. It will be a raw predicted value which must be
# post-processed to get the final correct predicted value.

# Make predictions using same iris input data. Even though it is the
# same dataset, it must be provided as an input parameter for the
# "predict" function.

predict(model,iris[,1:4])

rm(model)
rm(p)
```

---

pmml.lm *Generate PMML for lm objects*

Description

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

Usage

```
## S3 method for class 'lm'
pmml(model, model.name="Linear_Regression_Model",
      app.name="Rattle/PMML",
      description="Linear Regression Model", copyright=NULL,
      transforms=NULL, unknownValue=NULL, dataset=NULL,
      weights=NULL, ...)
```

Arguments

model	a lm object.
model.name	a name to be given to the model in the PMML code.
app.name	the name of the application that generated the PMML code.
description	a descriptive text for the Header element of the PMML code.
copyright	the copyright notice for the model.
transforms	data transformations represented in PMML via **pmmlTransformations**.
unknownValue	value to be used as the 'missingValueReplacement' attribute for all MiningFields.
dataset	the orginal training dataset, if available.
weights	the weights used for building the model.
...	further arguments passed to or from other methods.

Details

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

Author(s)

<rguha@indiana.edu>

References

R project: ***Fitting Linear Models***
http://stat.ethz.ch/R-manual/R-devel/library/stats/html/lm.html

Examples

```
fit <- lm(Sepal.Length ~ ., data=iris)
pmml(fit)

rm(fit)
```

---

pmml.multinom *Generate PMML for multinom objects*

Description

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

Usage

```
## S3 method for class 'multinom'
pmml(model, model.name="multinom_Model", app.name="Rattle/PMML",
     description="Multinomial Logistic Model", copyright = NULL,
     transforms = NULL, unknownValue=NULL, ...)
```

Arguments

model	a multinom object.
model.name	a name to be given to the model in the PMML code.
app.name	the name of the application that generated the PMML code.
description	a descriptive text for the Header element of the PMML code.
copyright	the copyright notice for the model.
transforms	data transformations represented in PMML via **pmmlTransformations**.
unknownValue	value to be used as the 'missingValueReplacement' attribute for all MiningFields.
...	further arguments passed to or from other methods.

Details

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

Author(s)

Zementis Inc.

References

R project CRAN package:
***nnet**: Feed-forward Neural Networks and Multinomial Log-Linear Models*
https://CRAN.R-project.org/package=nnet

---

pmml.naiveBayes *Generate PMML for naiveBayes objects*

Description

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

Usage

```
## S3 method for class 'naiveBayes'
pmml(model, model.name="naiveBayes_Model",
      app.name="Rattle/PMML", description="NaiveBayes Model",
      copyright=NULL, transforms=NULL, unknownValue=NULL,
      predictedField, ...)
```

Arguments

`model`	a `naiveBayes` object.
`model.name`	a name to be given to the model in the PMML code.
`app.name`	the name of the application that generated the PMML code.
`description`	a descriptive text for the Header element of the PMML code.
`copyright`	the copyright notice for the model.
`transforms`	data transformations represented in PMML via **pmmlTransformations**.
`unknownValue`	value to be used as the 'missingValueReplacement' attribute for all MiningFields.
`predictedField`	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.

Author(s)

Zementis Inc.

References

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

```
# Build a simple Naive Bayes model

# Upload the required library
library(e1071)
library(pmml)

# download an example dataset
data(houseVotes84)
house <- na.omit(houseVotes84)

# Construct an example model defining a threshold value of 0.003
model<-naiveBayes(Class~V1+V2+V3,data=house,threshold=0.003)

# Output the PMML representation
pmml(model,dataset=house,predictedField="Class")

rm(model)
```

`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 = "Rattle/PMML", description = "K Nearest Neighbors Model",
  copyright = NULL, transforms = NULL, unknownValue = NULL, ...)
```

Arguments

`model`	a neighbr object.
`model.name`	a name to be given to the model in the PMML code.
`app.name`	the name of the application that generated the PMML code.
`description`	a descriptive text for the Header element of the PMML code.
`copyright`	the copyright notice for the model.
`transforms`	data transformations represented in PMML via **pmmlTransformations**.
`unknownValue`	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

```
# continuous features with continuous target, categorical target,
# and neighbor ranking

## Not run:
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 or 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")

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")

pmml(fit)

## End(Not run)
```

---

pmml.nnet *Generate PMML for nnet objects*

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="Rattle/PMML",
      description="Neural Network PMML Model",
      copyright=NULL, transforms=NULL, unknownValue=NULL,
      ...)
```

Arguments

model	a nnet object.
model.name	a name to be given to the model in the PMML code.
app.name	the name of the application that generated the PMML code.
description	a descriptive text for the Header element of the PMML code.
copyright	the copyright notice for the model.
transforms	data transformations represented in PMML via **pmmlTransformations**.
unknownValue	value to be used as the 'missingValueReplacement' attribute for all MiningFields.
...	further arguments passed to or from other methods.

Details

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

Author(s)

Zementis Inc.

References

R project CRAN package:
***nnet**: Feed-forward Neural Networks and Multinomial Log-Linear Models*
https://CRAN.R-project.org/package=nnet

Examples

```
library(nnet)
fit <- nnet(Species ~ ., data=iris, size=4)
pmml(fit)

rm(fit)
```

---

pmml.randomForest *Generate PMML for randomForest objects*

Description

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

Usage

```
## S3 method for class 'randomForest'
pmml(model, model.name="randomForest_Model",
      app.name="Rattle/PMML",
      description="Random Forest Tree Model",
      copyright=NULL, transforms=NULL, unknownValue=NULL,
      parentInvalidValueTreatment="returnInvalid",
      childInvalidValueTreatment="asIs", ...)
```

Arguments

model	a randomForest object.
model.name	a name to be given to the model in the PMML code.
app.name	the name of the application that generated the PMML code.
description	a descriptive text for the Header element of the PMML code.
copyright	the copyright notice for the model.
transforms	data transformations represented in PMML via **pmmlTransformations**.
unknownValue	value to be used as the 'missingValueReplacement' attribute for all MiningFields.
parentInvalidValueTreatment	invalid value treatment at the top MiningField level.
childInvalidValueTreatment	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. The model will include not just the forest but also any pre-processing applied to the training data.

Author(s)

Zementis Inc.

References

R project CRAN package:
***randomForest**: Breiman and Cutler's random forests for classification and regression*
https://CRAN.R-project.org/package=randomForest

Examples

```
# Build a simple randomForest model

library(randomForest)
iris.rf <- randomForest(Species ~ ., data=iris, ntree=20)
```

```
# Convert to pmml

pmml(iris.rf)

rm(iris.rf)
```

`pmml.rfsrc` *Generate PMML for rfsrc objects*

Description

Generate the PMML representation for a **randomSurvivalForest** forest object.

Usage

```
## S3 method for class 'rfsrc'
pmml(model, model.name="rsf_Model",
      app.name="Rattle/PMML",
      description="Random Survival Forest Model",
      copyright=NULL, transforms=NULL, unknownValue=NULL, ...)
```

Arguments

`model`	a `forest` object contained in an object of class **randomSurvivalForest**, as that contained in the object returned by the function `rfsrc` with the parameter "forest=TRUE".
`model.name`	a name to be given to the model in the PMML code.
`app.name`	the name of the application that generated the PMML code.
`description`	a descriptive text for the Header of the PMML code.
`copyright`	the copyright notice for the model.
`transforms`	data transformations represented in PMML via **pmmlTransformations**.
`unknownValue`	value to be used as the 'missingValueReplacement' attribute for all MiningFields.
`...`	further arguments passed to or from other methods.

Details

This function is used to export the geometry of the forest to other PMML compliant applications, including graphics packages that are capable of printing binary trees. In addition, the user may wish to save the geometry of the forest for later retrieval and prediction on new data sets using `pmml.rfsrc` together with `pmml_to_rsf`.

The pmml package supports randomSurvivalForest up to version 2.5.0.

Author(s)

Zementis Inc.

References

- H. Ishwaran, U.B. Kogalur, E.H. Blackstone, M.S. Lauer (2008), /emphRANDOM SURVIVAL FORESTS. The Annals of Applied Statistics, Vol. 2, No. 3, 841-860
- H. Ishwaran and Udaya B. Kogalur (2006). Random Survival Forests. *Cleveland Clinic Technical Report*.

Examples

```
## Not run:
 # Works with randomForestSRC version 2.5.0.
 # library(randomForestSRC)
 # data(veteran)
 # veteran.out <- rfsrc(Surv(time, status)~., data = veteran, ntree = 5,
                      forest = TRUE, membership = TRUE)
 # pmml(veteran.out)

## End(Not run)
```

---

pmml.rpart *Generate PMML for rpart objects*

Description

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

Usage

```
## S3 method for class 'rpart'
pmml(model, model.name="RPart_Model",
      app.name="Rattle/PMML",
      description="RPart Decision Tree Model",
      copyright=NULL, transforms=NULL,
      unknownValue=NULL, dataset=NULL, ...)
```

Arguments

model	a rpart object.
model.name	a name to be given to the model in the PMML code.
app.name	the name of the application that generated the PMML code.
description	a descriptive text for the Header element of the PMML code.
copyright	the copyright notice for the model.
transforms	data transformations represented in PMML via **pmmlTransformations**.

unknownValue	value to be used as the 'missingValueReplacement' attribute for all MiningFields.
dataset	the original dataset used to train the model.
...	further arguments passed to or from other methods.

Details

The pmml function supports regression tree as well as classification tree of a rpart object. The object is represented in the PMML TreeModel format.

Author(s)

<Graham.Williams@togaware.com>, Zementis Inc.

References

R project CRAN package: ***rpart****: Recursive Partitioning*
https://CRAN.R-project.org/package=rpart

Examples

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

rm(fit)
```

---

pmml.rules *Generate PMML for arules objects*

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="Rattle/PMML",
      description="arules association rules model",
      copyright=NULL, transforms = NULL, ...)
## S3 method for class 'itemsets'
pmml(model, model.name="arules_Model",
      app.name="Rattle/PMML",
      description="arules frequent itemsets model",
      copyright=NULL, transforms = NULL, ...)
```

Arguments

model	a rules or itemsets object.
model.name	a name to be given to the model in the PMML code.
app.name	the name of the application that generated the PMML code.
description	a descriptive text for the Header element of the PMML code.
copyright	the copyright notice for the model.
transforms	not used in present version.
...	further arguments passed to or from other methods.

Details

The model is represented in the PMML AssociationModel format.

Author(s)

Michael Hahsler (<michael@hahsler.net>)

References

R project CRAN package: ***arules**: Mining Association Rules and Frequent Itemsets*
https://CRAN.R-project.org/package=arules

---

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 = "R-PMML", description = "Support Vector Machine Model",
  copyright = NULL, transforms = NULL, unknownValue = NULL,
  dataset = NULL, ...)
```

Arguments

model	an svm object from package **e1071**.
model.name	a name to be given to the model in the PMML code.
app.name	the name of the application that generated the PMML code.
description	a descriptive text for the Header element of the PMML code.
copyright	the copyright notice for the model.
transforms	data transformations represented in PMML via **pmmlTransformations**.
unknownValue	value to be used as the 'missingValueReplacement' attribute for all MiningFields.
dataset	required for one-classification only; data used to train one-class SVM model.
...	further arguments passed to or from other methods.

Details

The model is represented in the PMML SupportVectorMachineModel format.

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.

For a one-classification svm (OCSVM) model, the PMML has three OutputField elements. The OutputField `anomaly` is a boolean value that conforms to the DMG definition of an anomaly detection model; this value is `TRUE` when an anomaly is detected. This value is the opposite of the prediction by the e1071 object, which predicts FALSE when an anomaly is detected; that is, the R svm model predicts whether an input is an inlier. 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.

For example, say that for an input of observations, 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. 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. Note that there is no sign flip 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.

Anomaly detection SVM models are not yet supported by DMG PMML schema version 4.3. The PMML produced by this exporter uses an extended schema (4.3Ext), and can be consumed by Zementis products.

Value

PMML representation of the svm object.

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* http://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")
pmml(fit)

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

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


## End(Not run)
```

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

Description

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

Usage

```
## S3 method for class 'xgb.Booster'
pmml(model, model.name = "xboost_Model",
  app.name = "R", description = "Extreme Gradient Boosting Model",
  copyright = NULL, transforms = NULL, inputFeatureNames = NULL,
  outputLabelName = NULL, outputCategories = NULL,
  xgbDumpFile = NULL, unknownValue = NULL,
  parentInvalidValueTreatment = "returnInvalid",
  childInvalidValueTreatment = "asIs", ...)
```

Arguments

model	an object created by the 'xgboost' function
model.name	optional; the model name.
app.name	optional; name where the model was created.
description	optional; description of the model.
copyright	optional; a copyright statement.
transforms	optional; any pre-processing information from the pmmlTransformations package.
inputFeatureNames	input variable names used in training the model
outputLabelName	name of the predicted field
outputCategories	possible values of the predicted field, for classification models.
xgbDumpFile	name of file saved using 'xgb.dump' function.
unknownValue	optional; a missing value replacement.
parentInvalidValueTreatment	invalid value treatment at the top MiningField level.
childInvalidValueTreatment	invalid value treatment at the model segment MiningField level.
...	further arguments passed to 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 environmant 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.

See Also

pmml, PMML Schema

Examples

```
# Standard example using the xgboost package example model
# make the xgboost model using xgb.DMatrix object as inputs
## Not run:
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")

## End(Not run)

# the input feature names for the xgb.DMatrix object can be extracted as colnames(train$data)
# the output field name and categories must be inferred. Looking at train$label informs us
# that the output categories are either 0 or 1. The name cannot be inferred and so will be
# given a name "prediction1" save the tree information required in an external file
## Not run:
xgb.dump(model1, "model1.dumped.trees")

## End(Not run)
# Now all requiredinput parameters are known:
## Not run:
pmml(model1,inputFeatureNames=colnames(train$data),outputLabelName="prediction1",
 outputCategories=c("0","1"),xgbDumpFile="model1.dumped.trees")

## End(Not run)
```

```
# use iris dataset to make a multinomial model
# input data as a matrix
## Not run:
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)

## End(Not run)

# The field names are easily extracted from the columnnames and the categories are converted to
# numeric format by xgboost.
# save the tree information file
## Not run:
xgb.dump(model2, "model2.dumped.trees")

pmml(model2,inputFeatureNames=colnames(as.matrix(iris[,1:4])),outputLabelName="Species",
outputCategories=c(1,2,3),xgbDumpFile="model2.dumped.trees")

## End(Not run)
```

`pmmlCanExport` *Can this installation export PMML variables (particularly transforms).*

Description

This function is designed to be overriden by other packages that implement PMML export, particularly of transformations.

Usage

```
pmmlCanExport(vname)
```

Arguments

vname	a variable name to check whether it is exportable.

Author(s)

<Graham.Williams@togaware.com>

See Also

pmml.

pmmltoc *Generate C code from a PMML object - dummy function*

Description

This is a dummy function that does nothing. Plugins for Rattle are starting to appear which implement this for specific environments. This is experimental.

Usage

```
pmmltoc(p, name=NULL, includePMML=TRUE, includeMetaData=TRUE,
        exportClass=TRUE)
```

Arguments

p	pmml.
name	a name to give to the model in the C code.
includePMML	include the actual PMML as comments.
includeMetaData	include model information as comments.
exportClass	whether to export class or probability.

Author(s)

<Graham.Williams@togaware.com>

See Also

pmml.

savePMML *saves a xml object as an external PMML file.*

Description

A created pmml object can be saved for more efficient further processing via this function.

Usage

```
savePMML(doc, name, version=4.3)
```

Arguments

doc	the XML model created in R.
name	the name of the external file where the XML is to be saved.
version	the PMML version number the model is compliant with.

Author(s)

<tridivesh.jena@softwareag.net>

Examples

```
## Not run:
 # make a sample model
 library(gbm)
 library(pmml)
 data(audit)

 mod<-gbm(Adjusted~.,data=audit[,-c(1,4,6,9,10,11,12)],n.trees=3,interaction.depth=4)
 pmod <- pmml(mod)
 # Save to an external file
 savePMML(pmod, "GBMModel.pmml")

## End(Not run)
```

Index