Package 'pmmlTransformations'

June 12, 2019

Type Package

Title Transforms Input Data from a PMML Perspective

Version 1.3.3

Author Tridivesh Jena, Wen Ching Lin, Dmitriy Bolotov

Maintainer Dmitriy Bolotov <rpmmlsupport@softwareag.com>

Suggests knitr, testthat, rmarkdown, pmml (<= 1.5.7)

Description Allows for data to be transformed before using it to construct models. Builds structures to allow functions in the PMML package to
output transformation details in addition to the model in the resulting PMML file. 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.

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

License GPL (>= 2.1)

NeedsCompilation no

VignetteBuilder knitr

RoxygenNote 6.1.1

Encoding UTF-8

Repository CRAN

Date/Publication 2019-06-11 22:20:04 UTC

R topics documented:

pmmlTransformations-package 2
DiscretizeXform 3

FunctionXform 7
Initialize 8
MapXform 9
MinMaxXform 12
NormDiscreteXform 14
RenameVar 15
WrapData 17
ZScoreXform 18

Index **20**

`pmmlTransformations-package`

Data Transformations for PMML output

Description

NOTE: The functions in this package have been merged into the package **pmml**, starting with **pmml** 2.0.0. **pmmlTransformations** still works with **pmml** up to and including version 1.5.7, but will not receive any more updates. The examples throughout this package have (commented-out) calls to functions from **pmml**; if using **pmmlTransformations**, please use **pmml** 1.5.7 or older.

This package reads in raw data and allows the user to perform various transformations on the input data. The user can then use the derived data to construct models and output the model together with data transformations in the Predictive Model Markup Language (PMML) format through the use of the **pmml** package.

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.

Details

The general methodology to use this package is to first wrap the data with the **WrapData** function and then perform all desired transformations. The model, in PMML format, including the information on the transformations executed, can then be output by calling the **pmml** function of the pmml package. The **pmml** function in this case has to be given an additional parameter, **transform**, as shown in the example below.

This package can also be used as a transformation generator; output just the transformations information instead of the whole pmml model. To do so, one has to call the pmml function with the WrapData output but pass in a null value as the model name. An example can be seen in the documentation for the **WrapData** function.

This package does not support boolean dataTypes; only numeric and string data are supported.

Author(s)

Tridivesh Jena, Zementis, Inc.

References

PMML home page

PMML page describing various possible data transformations

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

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*.

Examples

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

# First create the wrapper object
   irisBox <- WrapData(iris)

# Perform a simple z-transformation on the first variable of the dataset:
# Sepal.Length. By default, the name of the transformed variable is
# "derived_Sepal.Length". The information of the transformation is added
# back to the wrapped data object.
   irisBox <- ZScoreXform(irisBox,"1")

# Build a simple lm model
   fit <- lm(Sepal.Width ~ derived_Sepal.Length + Petal.Length,
             data=irisBox$data)

# One may now output the model in PMML format using the command below.
# The PMML file will now include the data transformations as well as
# the model.
   # library(pmml)
   # fit_pmml <- pmml(fit, transform=irisBox)
```

DiscretizeXform — *Discretizes a continuous variable to a discrete one as indicated by interval mappings. This is in accordance to the PMML element:* ***Discretize***

Description

Creates a discrete variable from a continuous one. The discrete variable value depends on which interval the continuous variable value lies in. 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.

Usage

```
DiscretizeXform(boxdata, xformInfo, table, defaultValue=NA,
                mapMissingTo=NA, ...)
```

Arguments

boxdata	the wrapper object obtained by using the WrapData function on the raw data.
xformInfo	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.
defaultValue	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 'xformInfo' is a list, this is a vector with each element corresponding to the corresponding list element.
mapMissingTo	value to be given to the transformed variable if the value of the input variable is missing. If 'xformInfo' 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

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:

xformInfo = "[InVar->OutVar][InType->OutType]", table="TableFileName", defaultValue="defVal", mapMissingTo="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, defaultValue and mapMissingTo 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
if(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 'defaultValue' and 'mapMissingTo' parameters should be vectors. The first element of each vector will correspond to the derived field defined in the 1st element of the 'xformInfo' list etc. Although somewhat more complicated, this method is designed to not require any external features. Further, once the intial 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, Zementis, Inc.

See Also

WrapData, pmml

Examples

```
# Load the pmmlTransformations package
    library(pmmlTransformations)
    library(pmml)
# First wrap the data
    irisBox <- WrapData(iris)

## Not run:
# We wish to 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.
  irisBox <- DiscretizeXform(irisBox,
              xformInfo="[Sepal.Length -> dsl][double -> string]",
              table="intervals.csv",mapMissingTo="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)))

# 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)
  irisBox<-DiscretizeXform(irisBox,xformInfo=t,defaultValue=def,
                           mapMissingTo=mis)

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

FunctionXform *Add a function transformation to a WrapData object.*

Description

Add a function transformation to a WrapData object.

Usage

```
FunctionXform(boxdata, origFieldName, newFieldName = "newField",
  newFieldDataType = "numeric", formulaText, mapMissingTo = NA)
```

Arguments

boxdata	wrapper object obtained by using the WrapData function on raw data
origFieldName	string specifying name(s) of the original data field(s) being used in the transformation
newFieldName	name of the new field created by the transformation
newFieldDataType	data type of the new field created by the transformation
formulaText	string expression specifying the transformation
mapMissingTo	value to be given to the transformed variable if the value of any input variable is missing

Details

Calculate the expression provided in formulaText for every row in the boxdata$data data frame. The formulaText argument must represent a valid R expression, and any functions used in formulaText 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 WrapData object.

Value

R object containing the raw data, the transformed data and data statistics. The data data frame will contain a new newFieldName column, and fieldData will contain a new newFieldName row.

Author(s)

Dmitriy Bolotov

See Also

WrapData

Examples

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

# Wrap the data
irisBox <- WrapData(iris)

# Perform a transform on the Sepal.Length field:
# the value is squared and then divided by 100
irisBox <- FunctionXform(irisBox,origFieldName="Sepal.Length",
                         newFieldName="Sepal.Length.Transformed",
                         formulaText="(Sepal.Length^2)/100")

# Combine two fields to create another new feature:
irisBox <- FunctionXform(irisBox,
                         origFieldName="Sepal.Width, Petal.Width",
                         newFieldName="Width.Sum",
                         formulaText="Sepal.Width + Sepal.Length")

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

# Create pmml from the fit
# library(pmml)
# fit_pmml <- pmml(fit, transform=irisBox)
```

Initialize *Initialize internal variables in a WrapData object*

Description

Internal function used by other functions in the package. Used to initialize parameters in the wrapped data object.

Usage

```
Initialize(inbox)
```

Arguments

inbox	Object returned by the WrapData function.

Details

This is an internal function used ony by the other functions of this package.

Value

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

Author(s)

Tridivesh Jena, Zementis, Inc.

See Also

WrapData

`MapXform` *Implements a map between discrete values in accordance to the PMML element:* ***MapValues***

Description

Maps 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.

Usage

```
MapXform(boxdata, xformInfo, table,
        defaultValue=NA, mapMissingTo=NA, ...)
```

Arguments

`boxdata`	the wrapper object obtained by using the WrapData function on the raw data.
`xformInfo`	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.
`defaultValue`	the default value to be given to the transformed variable. If 'xformInfo' is a list, this is a vector with each element corresponding to the corresponding list element.
`mapMissingTo`	value to be given to the transformed variable if the value of the input variable is missing. If 'xformInfo' 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

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:

xformInfo = "[InVar1,InVar2,...->OutVar][InType1,InType2,...->OutType]",
table="TableFileName", defaultValue="defVal", mapMissingTo="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, defaultValue and mapMissingTo 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 'defaultValue' and 'mapMissingTo' parameters should be vectors. The first element of each vector will correspond to the derived field defined in the 1st element of the 'xformInfo' 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, Zementis, Inc.

See Also

WrapData, pmml

Examples

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

# First wrap the data
   auditBox <- WrapData(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 becreated, say "MapGender.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 = "0"
#
# Give "d_sex" the value 0 if the input variable value is
# missing.
  auditBox <- MapXform(auditBox,
                       xformInfo="[Sex -> d_sex][string->integer]",
                     table="MapGender.csv",mapMissingTo="0")

## End(Not run)
# same as above, with an extra variable, but using data frames.
# The top 2 rows gives the variable names and their data types.
# The rest represent the map. So 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))
   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.
   auditBox<-MapXform(auditBox,xformInfo=t,defaultValue=c(3),
                       mapMissingTo="2")

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

`MinMaxXform` *Normalizes continuous values in accordance to the PMML element:* ***NormContinuous***

Description

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

Usage

```
MinMaxXform(boxdata, xformInfo=NA, mapMissingTo=NA, ...)
```

Arguments

`boxdata`	the wrapper object obtained by using the WrapData function on the raw data.
`xformInfo`	specification of details of the transformation.
`mapMissingTo`	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 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 **MinMaxXform** command including all the optional parameters is in the format:

xformInfo="InputVar -> OutputVar[low_limit,high_limit], mapMissingTo="missingVal"

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.

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, Zementis, Inc.

See Also

WrapData

Examples

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

# First wrap the data
   irisBox <- WrapData(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.
   MinMaxXform(irisBox)

# Normalize the 1st column values of the dataset (Sepal.Length) to lie
# between 0 and 1 and give the derived variable the name "dsl"
   MinMaxXform(irisBox,xformInfo="column1 -> dsl")

# Repeat the above operation; adding the new transformed variable to
# the irisBox object
   irisBox <- MinMaxXform(irisBox,xformInfo="column1 -> dsl")

# Transform Sepal.Width(the 2nd column)
# The new transformed variable will be given the default name
# "derived_Sepal.Width"
   MinMaxXform(irisBox,xformInfo="column2")

# Repeat the same operation as above, this time using the variable name
   MinMaxXform(irisBox,xformInfo="Sepal.Width")

# Repeat the same operation as above, assign the transformed variable,
# "derived_Sepal.Width". the value of 0.5 if the input value of the
# "Sepal.Width" variable is missing
   MinMaxXform(irisBox,xformInfo="Sepal.Width", "mapMissingTo=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"
   MinMaxXform(irisBox,xformInfo="column2->[2,3]")

# Repeat the above transformation, this time the transformed variable
# lies between 0 and 10
   irisBox <- MinMaxXform(irisBox,xformInfo="column2->[,10]")
```

`NormDiscreteXform` *Normalize discrete values in accordance to the PMML element:* ***NormDiscrete***

Description

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.

Usage

```
NormDiscreteXform(boxdata, xformInfo=NA,
                  inputVar=NA, mapMissingTo=NA, ...)
```

Arguments

`boxdata`	the wrapper object obtained by using the WrapData function on the raw data.
`xformInfo`	specification of details of the transformation: the name of the input variable to be transformed.
`inputVar`	the input variable name in the data on which the transformation is to be applied
`mapMissingTo`	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 an input variable, **InputVar** and **missingVal**, the desired value of the transformed variable if the input variable value is missing, the NormDiscreteXform command including all optional parameters is in the format:

xformInfo="inputVar=input_variable, mapMissingTo=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 **xformInfo** and **inputVar** 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 **inputVar** 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 **InputVar_val1**, **InputVar_val2**, ...

Value

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

Author(s)

Tridivesh Jena, Zementis, Inc.

See Also

WrapData

Examples

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

# First wrap the data
   irisBox <- WrapData(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

  irisBox <- NormDiscreteXform(irisBox,inputVar="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.

  irisBox <- WrapData(iris)
  irisBox <- NormDiscreteXform(irisBox,xformInfo="Species")
```

RenameVar *Renames a variable in the WrapData transform object*

Description

Renames a variable inside a WrapData object

Usage

```
RenameVar(boxdata, xformInfo=NA, ...)
```

Arguments

boxdata	wrapper object obtained by using the WrapData function on the raw data.
xformInfo	specification of details of the renaming.
...	further arguments passed to or from other methods.

Details

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

xformInfo="InputVar -> OutputVar"

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 renamings performed.

Value

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

Author(s)

Tridivesh Jena, Zementis, Inc.

See Also

WrapData.

Examples

```
# Load the standard iris dataset, already built into R
   data(iris)

# First wrap the data
   irisBox <- WrapData(iris)

# We wish to refer to the variables "Sepal.Length" and
# "Sepal.Width" as "SL" and "SW"
  irisBox <- RenameVar(irisBox,"column1->SL")
  irisBox <- RenameVar(irisBox,"Sepal.Width->SW")
```

`WrapData` *Wrap raw data in an R object*

Description

Wrap raw data read in a R object. This object is then read in by the transform functions and the data in it is transformed.

Usage

```
WrapData(indata, useMatrix=FALSE)
```

Arguments

`indata`	the raw data set.
`useMatrix`	boolean value indicating whether data should be stored in matrix format as well.

Details

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 'useMatrix' 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, Zementis, Inc.

See Also

`pmml`

Examples

```
# Load the standard iris dataset, already built into R
   data(iris)

# Make a wrapper object for the iris dataset to use with
# transformation functions
   irisBox <- WrapData(iris)
```

```
# Output only the transformations in PMML format.
# This example will output just an empty "LocalTransformations"
# element as no transformations were performed.
   # library(pmml)
   # pmml(NULL,transforms=irisBox)
# This will also work
   # pmml(,transforms=irisBox)
```

`ZScoreXform` *Performs a z-score normalization on continuous values in accordance to the PMML element:* ***NormContinuous***

Description

Performs a z-score normalization on data given in WrapData format

Usage

```
ZScoreXform(boxdata, xformInfo=NA, mapMissingTo=NA, ...)
```

Arguments

`boxdata`	wrapper object obtained by using the WrapData function on the raw data.
`xformInfo`	specification of details of the transformation.
`mapMissingTo`	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 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 ZScoreXform command including all the optional parameters is:

xformInfo="InputVar -> OutputVar", mapMissingTo="missingVal"

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.
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, Zementis, Inc.

See Also

WrapData.

Examples

```
# Load the standard iris dataset, already built into R
   data(iris)

# First wrap the data
   irisBox <- WrapData(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
   ZScoreXform(irisBox)

# Perform a z-transform on the 1st column of the dataset (Sepal.Length)
# and give the derived variable the name "dsl"
   ZScoreXform(irisBox,xformInfo="column1 -> dsl")

# Repeat the above operation; adding the new transformed variable
# to the irisBox object
   irisBox <- ZScoreXform(irisBox,xformInfo="column1 -> dsl")

# Transform Sepal.Width(the 2nd column)
# The new transformed variable will be given the default name
# "derived_Sepal.Width"
   ZScoreXform(irisBox,xformInfo="column2")

# Repeat the same operation as above, this time using the variable
# name
   ZScoreXform(irisBox,xformInfo="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
# irisBox object.
   irisBox <- ZScoreXform(irisBox,xformInfo="Sepal.Width",
                          "mapMissingTo=1.0")
```

Index

∗Topic **manip**
DiscretizeXform, 3
MapXform, 9
MinMaxXform, 12
NormDiscreteXform, 14
RenameVar, 15
ZScoreXform, 18
∗Topic **methods**
RenameVar, 15
ZScoreXform, 18
∗Topic **pmml**
pmmlTransformations-package, 2
∗Topic **utilities**
RenameVar, 15
ZScoreXform, 18

DiscretizeXform, 3

FunctionXform, 7

Initialize, 8

MapXform, 9
MinMaxXform, 12

NormDiscreteXform, 14

pmml, *5*, *10*, *17*
pmmlTransformations
*(*pmmlTransformations-package*)*, 2
pmmlTransformations-package, 2

RenameVar, 15

WrapData, *5*, *8–10*, *13*, *15*, *16*, 17, *19*

ZScoreXform, 18