Package 'healthcareai'

May 5, 2017

Type Package

Title Tools for Healthcare Machine Learning

Version 0.1.12

Date 2017-05-03

Description A machine learning toolbox tailored to healthcare data.
Aids in data cleaning, model development, hyperparameter tuning, and model deployment in a production SQL environment. Algorithms currently supported are Lasso, Random Forest, and Linear Mixed Model.

License MIT + file LICENSE

LazyData TRUE

Depends R (>= 3.2.3)

Imports caret, data.table, DBI, doParallel, e1071, grpreg, lme4, odbc, pROC, R6, ranger, ROCR, RODBC, RSQLite

RoxygenNote 6.0.1

Suggests testthat

URL http://healthcareai-r.readthedocs.io

BugReports https://github.com/HealthCatalystSLC/healthcareai-r/issues

NeedsCompilation no

Author Levi Thatcher [aut, cre],
Mike Mastanduno [aut],
Taylor Miller [aut],
Taylor Larsen [aut]

Maintainer Levi Thatcher <levi.thatcher@healthcatalyst.com>

Repository CRAN

Date/Publication 2017-05-05 03:11:36 UTC

R topics documented:

`calculateAllCorrelations` *Correlation analysis on an input table over all numeric columns*

Description

Calculate correlations between every numeric column in a table

Usage

```
calculateAllCorrelations(df)
```

Arguments

`df`	A data frame

Value

A data frame with column names and corresponding correlations with the target column

References

http://healthcare.ai

See Also

healthcareai

Examples

```
df <- data.frame(a=c(1,2,3,4,5,6),
b=c(6,5,4,3,2,1),
c=c(3,4,2,1,3,5),
d=c('M','F','F','F','M','F')) #<- is ignored

dfResult <- calculateAllCorrelations(df)
dfResult
```

calculateHourBins *Calculate a vector of reasonable time bins*

Description

Given a number of hours, generate a reasonable vector of bins in hours such that the first day is divided into multiple days are divided into 24 h bins up to 90 days worth. Typically used with featureAvailabilityProfiler

Usage

```
calculateHourBins(lastHourOfInterest)
```

Arguments

lastHourOfInterest	Number (hour) scalar representing the last hour of interest

Value

numeric vector of hours, reasonably spaced

References

http://healthcare.ai

See Also

healthcareai result <- calculateHourBins(90) result

calculatePerformance *Generate performance metrics after model has been trained*

Description

Generates AU_ROC and AU_PR (including 95

Usage

```
calculatePerformance(predictions, ytest, type)
```

Arguments

predictions	A vector of predictions from a machine learning model.
ytest	A vector of the true labels. Must be the same length as predictions.
type	A string. Indicates model type and can be "regression" or "classification". Defaults to SS.

Value

Curves (if classification); otherwise nothing. Prints results.

References

http://healthcare.ai

See Also

healthcareai

`calculateSDChanges` *Calculate std deviation up/down for each numeric field in row*

Description

Add/subtract each numeric col (for each row) by std dev, such that we have a new alternate data frame

Usage

```
calculateSDChanges(dfOriginal, rowNum, numColLeaveOut, sizeOfSDPerturb = 0.5)
```

Arguments

`dfOriginal`	Data frame from Error in as.double(y) : cannot coerce type 'S4' to vector of type 'double' which we'll draw a row for alt-scenarios
`rowNum`	Row in dfOriginal that we'll create alt-scenarios for
`numColLeaveOut`	Numeric columns to leave out of alterlative scenarios
`sizeOfSDPerturb`	Default is 0.5. Shrink or expand SD drop/addition

References

http://healthcare.ai

See Also

healthcareai

Examples

```
df <- data.frame(a=c(1,2,3),
                 b=c('m','f','m'),
                 c=c(0.7,1.4,2.4),
                 d=c(100,250,200),
                 e=c(400,500,505))


dfResult <- calculateSDChanges(dfOriginal = df,
                               rowNum = 2,
                               numColLeaveOut = c('d','e'),
                               sizeOfSDPerturb = 0.5)
dfResult
```

`calculateTargetedCorrelations` *Correlation analysis on an input table, focusing on one target variable*

Description

Calculates correlations between each numeric column in a table and and a target column

Usage

```
calculateTargetedCorrelations(df, targetCol)
```

Arguments

`df`	A data frame
`targetCol`	Name of target column against which correlations will be calculated

Value

A data frame with column names and corresponding correlations and p-values with the target column

References

http://healthcare.ai

See Also

healthcareai

Examples

```
df <- data.frame(a=c(1,2,3,4,5,6),
b=c(6,5,4,3,2,1),
c=c(3,4,2,1,3,5),
d=c('M','F','F','F','M','F')) #<- is ignored

dfResult <- calculateTargetedCorrelations(df=df,targetCol='c')
dfResult
```

`calulcateAlternatePredictions` *Recalculate predicted value based on alternate scenarios*

Description

After getting alternate features via calculateSDChanges recalculate predicted values for each row in df.

Usage

```
calulcateAlternatePredictions(df, modelObj, type, outVectorAppend = NULL,
  removeCols = NULL)
```

Arguments

`df`	Data frame from which we'll calculate alternate predictions
`modelObj`	Object representing the model that is used for predictions
`type`	String representing which type of model is used
`outVectorAppend`	Optional list of values that we'll append predictions to. If not used, then a new vector is created.
`removeCols`	Optional list of column names to remove before calculating alternate predictions.

References

http://healthcare.ai

See Also

healthcareai

Examples

```
library(caret)
df <- data.frame(a=c(1,2,3,1),
                 b=c('m','f','m','m'),
                 c=c(0.7,1.4,2.4,2.0),
                 d=c(100,250,200,150))

# Get alternate feature scenarios
dfResult <- calculateSDChanges(df=df,
                               rowNum=2,
                               sizeOfSDPerturb = 0.5,
                               numColLeaveOut='d')

y <- c('y','n','y','n')

# Train model on original data frame
glmOb <- train(x = df,y = y,method = 'glm',family = 'binomial')

outList <- calulcateAlternatePredictions(df=dfResult,
                                         modelObj=glmOb,
                                         type='lasso')
outList
```

---

convertDateTimeColToDummies

Convert datetime column into dummy columns

Description

Convert datetime column into dummy columns of day, hour, etc, such that one can use daily and seasonal patterns in their model building.

Usage

```
convertDateTimeColToDummies(df, dateTimeCol, depth = "h",
  returnDtCol = FALSE)
```

Arguments

df	A data frame. Indicates the datetime column.
dateTimeCol	A string. Column name in df that will be converted into several columns.
depth	A string. Specifies the depth with which to expand extra columns (starting with a year column). 'd' expands to day, 'h' expands to hour (default), m' expands to minute, and 's' expands to second.
returnDtCol	A boolean. Return the original dateTimeCol with the modified data frame?

Value

A data frame which now includes several columns based on time rather than just one datetime column

References

http://healthcare.ai

See Also

healthcareai

Examples

```
dtCol <- c('2001-06-09 12:45:05','2002-01-29 09:30:05','2002-02-02 07:36:50',
           '2002-03-04 16:45:01','2002-11-13 20:00:10','2003-01-29 07:31:43',
           '2003-07-07 17:30:02','2003-09-28 01:03:20')
y1 <- c(.5,1,3,6,8,13,14,1)
y2 <- c(.8,1,1.2,1.2,1.2,1.3,1.3,1)
df <- data.frame(dtCol,y1,y2)

df <- convertDateTimeColToDummies(df, 'dtCol')
head(df)
```

---

countDaysSinceFirstDate

Creates column based on days since first date

Description

Adds a new column to the data frame, which shows days since first day in input column

Usage

```
countDaysSinceFirstDate(df, dtCol, returnDtCol = FALSE)
```

Arguments

df	A data frame
dtCol	A string denoting the date-time column of interest
returnDtCol	A boolean. Return the original dtCol with the modified data frame?

Value

A data frame that now has a new column

References

http://healthcare.ai

See Also

healthcareai

Examples

```
dtCol = c('2001-06-09 12:45:05','2002-01-29 09:30:05','2002-02-02 07:36:50',
'2002-03-04 16:45:01','2002-11-13 20:00:10','2003-01-29 07:31:43',
'2003-07-07 17:30:02','2003-09-28 01:03:20')
y1 <- c(.5,1,3,6,8,13,14,1) # Not being used at all
df <- data.frame(dtCol, y1)
head(df)
dfResult <- countDaysSinceFirstDate(df, 'dtCol')
head(dfResult)
```

---

countPercentEmpty *DEPRECATED. Calculates percentage of each column in df that is NULL (NA)*

Description

Returns a vector with percentage of each column that is NULL in the original data frame

Usage

```
countPercentEmpty(df)
```

Arguments

df	A data frame

Value

A vector that contains the percentage of NULL in each column

References

http://healthcare.ai

See Also

healthcareai

Examples

```
df = data.frame(a=c(1,2,NA,NA,3),
                b=c(NA,NA,NA,NA,NA),
                c=c(NA,NA,'F','M',NA))
colList = countPercentEmpty(df)
colList
```

`featureAvailabilityProfiler`

Calculate and plot data availability over time

Description

Helps you determine how much data is present in each feature, by hour, after a particular event (like patient admit)

Usage

```
featureAvailabilityProfiler(df, startDateColumn = "AdmitDTS",
  lastLoadDateColumn = "LastLoadDTS", plotProfiler = TRUE)
```

Arguments

`df`	A dataframe
`startDateColumn`	Optional string of the column name, representing the date of the starting event of interest (e.g., patient admit)
`lastLoadDateColumn`	Optional string of the column name, representing the date the row was loaded into the final dataset (i.e., via daily ETL)
`plotProfiler`	Default is TRUE. Whether to plot profiler results

Value

a list, that has as many vectors as columns in the original dataframe, with each vector holding the percentage full for each hour

References

http://healthcare.ai

See Also

healthcareai

Examples

```
df <- data.frame(a = c(2,1,3,5,4,NA,7,NA),
                 b = c(0.7,-2,NA,-4,-5,-6,NA,NA),
                 c = c(100,300,200,NA,NA,NA,NA,500),
                 d = c(407,500,506,504,NA,NA,NA,405),
                 admit = c('2012-01-01 00:00:00','2012-01-01 00:00:00',
                           '2012-01-01 12:00:00','2012-01-01 12:00:00',
                           '2012-01-02 00:00:00','2012-01-02 00:00:00',
                           '2012-01-02 12:00:00','2012-01-02 12:00:00'),
                 loaded = c('2012-01-03 00:00:00','2012-01-03 00:00:00',
                            '2012-01-03 00:00:00','2012-01-03 00:00:00',
                            '2012-01-03 00:00:00','2012-01-03 00:00:00',
                            '2012-01-03 00:00:00','2012-01-03 00:00:00'))
str(df)
head(df)

d <- featureAvailabilityProfiler(df = df,
                                 startDateColumn = 'admit',
                                 lastLoadDateColumn = 'loaded')
d # Look at the data in the console
```

`findBestAlternateScenarios` *Find most biggest drop in predictive probability across alternate features*

Description

Compare each alternate probability prediction and determine which ones are lowest compared to the original; return the top three column names that lead to the biggest drop and their target value.

Usage

```
findBestAlternateScenarios(dfAlternateFeat, originalRow, predictionVector,
  predictionOriginal)
```

Arguments

`dfAlternateFeat`	Data frame of alternate feature values
`originalRow`	Row from original data frame upon which alternates are based
`predictionVector`	List of alternate predictions
`predictionOriginal`	Scalar representing original prediction for row, without alternative scenario

References

http://healthcare.ai

See Also

healthcareai

Examples

```
library(caret)
df <- data.frame(a = c(1,2,3,1),
                 b = c('m','f','m','m'),
                 c = c(0.7,1.4,2.4,2.0),
                 d = c(100,250,200,150))

y <- c('y','n','y','n')

dfAlt <- calculateSDChanges(df = df,
                            rowNum = 2,
                            sizeOfSDPerturb = 0.5,
                            numColLeaveOut = 'd')

glmOb <- train(x = df,y = y,method = 'glm',family = 'binomial')

originalPred <- predict(object = glmOb,
                        newdata = df[4,],
                        type = 'prob')

alternatePred <- calulcateAlternatePredictions(df = dfAlt,
                                               modelObj = glmOb,
                                               type = 'lasso',
                                               removeCols = 'AlteredCol')

dfResult <- findBestAlternateScenarios(dfAlternateFeat = dfAlt,
                                       originalRow = df[4,],
                                       predictionVector = as.numeric(alternatePred),
                                       predictionOriginal = originalPred[[2]])

dfResult
```

`findTrends` *Find any columns that have a trend above a particular threshold*

Description

Find numeric columns in data frame that have an absolute slope greater than that specified via threshold argument.

Usage

```
findTrends(df, dateCol, groupbyCol)
```

Arguments

df	A data frame
dateCol	A string denoting the date column
groupbyCol	A string denoting the column by which to group

Value

A data frame containing the dimensional attribute (ie gender), the subset the data was grouped by (ie M/F), the measures that had trends (ie, mortality or readmission), and the ending month.

References

http://healthcare.ai

See Also

healthcareai

Examples

```
dates <- c(as.Date('2012-01-01'),as.Date('2012-01-02'),as.Date('2012-02-01'),
          as.Date('2012-03-01'),as.Date('2012-04-01'),as.Date('2012-05-01'),
          as.Date('2012-06-01'),as.Date('2012-06-02'))
y1 <- c(0,1,2,6,8,13,14,16)               # large positive
y2 <- c(.8,1,1.2,1.2,1.2,1.3,1.3,1.5)     # small positive
y3 <- c(1,0,-2,-2,-4,-5,-7,-8)            # big negative
y4 <- c(.5,0,-.5,-.5,-.5,-.5,-.6,0)       # small negative
gender <- c('M','F','F','F','F','F','F','F')
df <- data.frame(dates,y1,y2,y3,y4,gender)

dfResult <- findTrends(df = df,
                       dateCol = 'dates',
                       groupbyCol = 'gender')
dfResult
```

---

generateAUC *Generate ROC or PR curve for a dataset.*

Description

Generates ROC curve and AUC for Sensitivity/Specificity or Precision/Recall.

Usage

```
generateAUC(predictions, labels, aucType = "SS", plotFlg = FALSE,
  allCutoffsFlg = FALSE)
```

Arguments

predictions	A vector of predictions from a machine learning model.
labels	A vector of the true labels. Must be the same length as predictions.
aucType	A string. Indicates AUC_ROC or AU_PR and can be "SS" or "PR". Defaults to SS.
plotFlg	Binary value controlling plots. Defaults to FALSE (no).
allCutoffsFlg	Binary value controlling list of all thresholds. Defaults to FALSE (no).

Value

AUC: A number between 0 and 1. Integral AUC of chosen plot type.

IdealCutoffs: Array of cutoff and associated TPR/FPR or pre/rec.

Performance: ROCR performance class containing all ROC information.

References

http://healthcare.ai

See Also

healthcareai

Examples

```
# generate data
# example probablities
df <- data.frame(a = rep( seq(0,1,by=0.1), times=9))
# example ground truth values
df[,'b'] <- (runif(99,0,1)*df[,'a']) > 0.5

# prepare vectors
pred <- df[,'a']
labels <- df[,'b']

# generate the AUC
auc = generateAUC(predictions = pred,
                  labels = labels,
                  aucType = 'SS',
                  plotFlg = TRUE,
                  allCutoffsFlg = TRUE)
```

getCutOffList *Function to return ideal cutoff and TPR/FPR or precision/recall.*

Description

Calculates ideal cutoff by proximity to corner of the ROC curve. Usually called from generateAUC

Usage

```
getCutOffList(perf, aucType = "SS", allCutoffsFlg = FALSE)
```

Arguments

perf	An ROCR performance class. (Usually made by generateAUC)
aucType	A string. Indicates AUC_ROC or AU_PR and can be "SS" or "PR". Defaults to SS.
allCutoffsFlg	Binary value controlling list of all thresholds.

Value

Array of ideal cutoff and associated TPR/FPR or pre/rec.

References

http://healthcare.ai

See Also

healthcareai

groupedLOCF *Last observation carried forward*

Description

Carries the last observed value forward for all columns in a data.table grouped by an id.

Usage

```
groupedLOCF(df, id)
```

Arguments

df	data frame sorted by an ID column and a time or sequence number column.
id	A column name (in ticks) in df to group rows by.

Value

A data frame where the last non-NA values are carried forward (overwriting NAs) until the group ID changes.

References

http://healthcare.ai

See Also

healthcareai

Examples

```
df <- data.frame(personID=c(1,1,2,2,3,3,3),
                 wt=c(.5,NA,NA,NA,.3,.7,NA),
                 ht=c(NA,1,3,NA,4,NA,NA),
                 date=c('01/01/2015','01/15/2015','01/01/2015','01/15/2015',
                        '01/01/2015','01/15/2015','01/30/2015'))

head(df,n=7)

dfResult <- groupedLOCF(df, 'personID')

head(dfResult, n = 7)
```

---

healthcareai *healthcareai: a streamlined way to develop and deploy models*

Description

healthcareai provides a clean interface that lets one create and compare multiple models on your data, and then deploy the model that is most accurate. healthcareai also includes functions for data exploration, data cleaning, and model evaluation.

Details

This is done in a four-step process:

1. **Load and profile data**
   Use selectData to pull data directly from the SQL database. Then, featureAvailabilityProfiler and countPercentEmpty can help determine how many null values are in a column and how they are populated over time. calculateTargetedCorrelations and findTrends can help explore data. Manipulate dates using orderByDate and countDaysSinceFirstDate.
2. **Develop a machine learning model**
   Use LassoDevelopment or RandomForestDevelopment and test different combinations of features. Determine the best model using:

- Area under the ROC curve or area under the Performance-Recall curve for classification problems (yes or no response).
- Mean squared error for regression problems (continuous response).

3. **Deploy the machine learning model**
   `LassoDeployment` or `RandomForestDeployment` to create a final model, automatically save it, predict against test data, and push predicted values into a SQL environment. This can be tested locally, but eventually lives on the production server.
4. **Monitor performance in production environment**
   After generating predictions and getting ground truth values, use `generateAUC` to monitor performance over time. This should happen after greater than 1000 predictions have been made or 30 days have passed.

References

http://healthcare.ai

See Also

`LinearMixedModelDevelopment`

`LinearMixedModelDeployment`

`RiskAdjustedComparisons`

`imputeColumn`

`groupedLOCF`

`selectData`

`writeData`

`orderByDate`

`isBinary`

`removeRowsWithNAInSpecCol`

`removeColsWithAllSameValue`

`returnColsWithMoreThanFiftyCategories`

`findTrends`

`convertDateTimeColToDummies`

`countDaysSinceFirstDate`

`calculateTargetedCorrelations`

`calculateAllCorrelations`

`featureAvailabilityProfiler`

`generateAUC`

`imputeColumn` *Perform imputation on a vector*

Description

This class performs imputation on a vector. For numeric vectors the vector-mean is used; for factor columns, the most frequent value is used.

Usage

```
imputeColumn(v)
```

Arguments

v	A vector, or column of values with NAs.

Value

A vector, or column of values now with no NAs

References

http://healthcare.ai

See Also

healthcareai

Examples

```
# For a numeric vector
vResult <- imputeColumn(c(1,2,3,NA))

# For a factor vector
vResult <- imputeColumn(c('Y','N','Y',NA))

# To use this function on an entire data frame:
df <- data.frame(a=c(1,2,3,NA),
                 b=c('Y','N','Y',NA))
df[] <- lapply(df, imputeColumn)
head(df)
```

`initializeParamsForTesting` *Function to initialize and populate the SupervisedModelDevelopmentParams each time a unit test is run.*

Description

Initialize and populate SupervisedModelDevelopmentParams

Usage

```
initializeParamsForTesting(df)
```

Arguments

`df`	A data frame to use with the new supervised model.

Value

Supervised Model Development Params class

References

http://healthcare.ai

See Also

`healthcareai`

`isBinary` *Check if a vector has only two unique values.*

Description

Check if a vector is binary (not counting NA's)

Usage

```
isBinary(v)
```

Arguments

`v`	A vector, or column of values

Value

A boolean

References

http://healthcare.ai

See Also

healthcareai

Examples

```
isBinary(c(1,2,NA))
isBinary(c(1,2,3))
```

`LassoDeployment` *Deploy a production-ready predictive Lasso model*

Description

This step allows one to

- Create a final model on all of your training data
- Automatically save the model
- Run the model against test data to generate predictions
- Push these predictions to SQL Server

Usage

```
LassoDeployment(type, df, grainCol, testWindowCol, predictedCol,
impute, debug)
```

Arguments

`type`	The type of model (either 'regression' or 'classification')
`df`	Dataframe whose columns are used for calc.
`grainCol`	The dataframe's column that has IDs pertaining to the grain
`testWindowCol`	Y or N. This column dictates the split between model training and test sets. Those rows with N in this column indicate the training set while those that have Y indicate the test set
`predictedCol`	Column that you want to predict. If you're doing classification then this should be Y/N.
`impute`	For training df, set all-column imputation to F or T. This uses mean replacement for numeric columns and most frequent for factorized columns. F leads to removal of rows containing NULLs.
`debug`	Provides the user extended output to the console, in order to monitor the calculations throughout. Use T or F.

Format

An object of class `R6ClassGenerator` of length 24.

See Also

healthcareai

Examples

```
#### Regression Example using csv data ####
ptm <- proc.time()
library(healthcareai)

# setwd('C:/Yourscriptlocation/Useforwardslashes') # Uncomment if using csv

# Can delete this line in your work
csvfile <- system.file("extdata",
                       "HCRDiabetesClinical.csv",
                       package = "healthcareai")

# Replace csvfile with 'path/file'
df <- read.csv(file = csvfile,
               header = TRUE,
               na.strings = c("NULL", "NA", ""))

head(df)
str(df)

# Remove unnecessary columns
df$PatientID <- NULL

p <- SupervisedModelDeploymentParams$new()
p$type <- "regression"
p$df <- df
p$grainCol <- "PatientEncounterID"
p$testWindowCol <- "InTestWindowFLG"
p$predictedCol <- "A1CNBR"
p$impute <- TRUE
p$debug <- FALSE
p$useSavedModel <- FALSE
p$cores <- 1
p$writeToDB <- FALSE

dL <- LassoDeployment$new(p)
dL$deploy()

df <- dL$getOutDf()
# Write to CSV (or JSON, MySQL, etc) using plain R syntax
# write.csv(df,'path/predictionsfile.csv')

print(proc.time() - ptm)
```

```
#### Classification example using SQL Server data ####
# This example requires you to first create a table in SQL Server
# If you prefer to not use SAMD, execute this in SSMS to create output table:
# CREATE TABLE dbo.HCRDeployClassificationBASE(
#   BindingID float, BindingNM varchar(255), LastLoadDTS datetime2,
#   PatientEncounterID int, <--change to match inputID
#   PredictedProbNBR decimal(38, 2),
#   Factor1TXT varchar(255), Factor2TXT varchar(255), Factor3TXT varchar(255)
# )

ptm <- proc.time()
library(healthcareai)

connection.string <- "
driver={SQL Server};
server=localhost;
database=SAM;
trusted_connection=true
"

query <- "
SELECT
 [PatientEncounterID] --Only need one ID column for random forest
,[SystolicBPNBR]
,[LDLNBR]
,[A1CNBR]
,[GenderFLG]
,[ThirtyDayReadmitFLG]
,[InTestWindowFLG]
FROM [SAM].[dbo].[HCRDiabetesClinical]
--no WHERE clause, because we want train AND test
"

df <- selectData(connection.string, query)

head(df)
str(df)

p <- SupervisedModelDeploymentParams$new()
p$type <- "classification"
p$df <- df
p$grainCol <- "PatientEncounterID"
p$testWindowCol <- "InTestWindowFLG"
p$predictedCol <- "ThirtyDayReadmitFLG"
p$impute <- TRUE
p$debug <- FALSE
p$useSavedModel <- FALSE
p$cores <- 1
p$sqlConn <- connection.string
p$destSchemaTable <- "dbo.HCRDeployClassificationBASE"
```

```
dL <- LassoDeployment$new(p)
dL$deploy()

print(proc.time() - ptm)



#### Regression Example using SQL Server data ####
# This example requires you to first create a table in SQL Server
# If you prefer to not use SAMD, execute this in SSMS to create output table:
# CREATE TABLE dbo.HCRDeployRegressionBASE(
#   BindingID float, BindingNM varchar(255), LastLoadDTS datetime2,
#   PatientEncounterID int, <--change to match inputID
#   PredictedValueNBR decimal(38, 2),
#   Factor1TXT varchar(255), Factor2TXT varchar(255), Factor3TXT varchar(255)
# )

ptm <- proc.time()
library(healthcareai)

connection.string <- "
driver={SQL Server};
server=localhost;
database=SAM;
trusted_connection=true
"

query <- "
SELECT
 [PatientEncounterID] --Only need one ID column for random forest
,[SystolicBPNBR]
,[LDLNBR]
,[A1CNBR]
,[GenderFLG]
,[ThirtyDayReadmitFLG]
,[InTestWindowFLG]
FROM [SAM].[dbo].[HCRDiabetesClinical]
--no WHERE clause, because we want train AND test
"

df <- selectData(connection.string, query)

head(df)
str(df)

p <- SupervisedModelDeploymentParams$new()
p$type <- "regression"
p$df <- df
p$grainCol <- "PatientEncounterID"
p$testWindowCol <- "InTestWindowFLG"
p$predictedCol <- "A1CNBR"
p$impute <- TRUE
p$debug <- FALSE
```

```
p$useSavedModel <- FALSE
p$cores <- 1
p$sqlConn <- connection.string
p$destSchemaTable <- "dbo.HCRDeployRegressionBASE"

dL <- LassoDeployment$new(p)
dL$deploy()

print(proc.time() - ptm)
```

`LassoDevelopment` *Compare predictive models, created on your data*

Description

This step allows you to create a Lasso model, based on your data.

Usage

```
LassoDevelopment(object, type, df, grainCol, predictedCol, impute,
debug)
```

Arguments

`object`	of SuperviseModelParameters class for $new() constructor
`type`	The type of model (either 'regression' or 'classification')
`df`	Dataframe whose columns are used for calc.
`grainCol`	Optional. The dataframe's column that has IDs pertaining to the grain. No ID columns are truly needed for this step.
`predictedCol`	Column that you want to predict. If you're doing classification then this should be Y/N.
`impute`	Set all-column imputation to F or T. This uses mean replacement for numeric columns and most frequent for factorized columns. F leads to removal of rows containing NULLs.
`debug`	Provides the user extended output to the console, in order to monitor the calculations throughout. Use T or F.

Format

An object of class `R6ClassGenerator` of length 24.

References

http://healthcare.ai

See Also

RandomForestDevelopment

LinearMixedModelDevelopment

healthcareai

Examples

```
#### Example using iris dataset ####
ptm <- proc.time()
library(healthcareai)

data(iris)
head(iris)

set.seed(42)

p <- SupervisedModelDevelopmentParams$new()
p$df <- iris
p$type <- "regression"
p$impute <- TRUE
p$grainCol <- ""
p$predictedCol <- "Sepal.Width"
p$debug <- FALSE
p$cores <- 1

# Run Lasso
lasso <- LassoDevelopment$new(p)
lasso$run()

set.seed(42)
# Run Random Forest
rf <- RandomForestDevelopment$new(p)
rf$run()

print(proc.time() - ptm)

#### Example using csv data ####
library(healthcareai)
# setwd('C:/Your/script/location') # Needed if using YOUR CSV file
ptm <- proc.time()

# Can delete this line in your work
csvfile <- system.file("extdata", "HCRDiabetesClinical.csv", package = "healthcareai")
# Replace csvfile with '/path/to/yourfile'
df <- read.csv(file = csvfile, header = TRUE, na.strings = c("NULL", "NA", ""))

head(df)

df$PatientID <- NULL
df$InTestWindowFLG <- NULL
```

```
set.seed(42)
p <- SupervisedModelDevelopmentParams$new()
p$df <- df
p$type <- "classification"
p$impute <- TRUE
p$grainCol <- "PatientEncounterID"
p$predictedCol <- "ThirtyDayReadmitFLG"
p$debug <- FALSE
p$cores <- 1

# Run Lasso
lasso <- LassoDevelopment$new(p)
lasso$run()

set.seed(42)
# Run Random Forest
rf <- RandomForestDevelopment$new(p)
rf$run()

print(proc.time() - ptm)


#### This example is specific to Windows and is not tested.
#### Example using SQL Server data #### This example requires: 1) That you alter
#### your connection string / query

ptm <- proc.time()
library(healthcareai)

connection.string <- "
driver={SQL Server};
server=localhost;
database=SAM;
trusted_connection=true
"

query <- "
SELECT
[PatientEncounterID]
,[SystolicBPNBR]
,[LDLNBR]
,[A1CNBR]
,[GenderFLG]
,[ThirtyDayReadmitFLG]
FROM [SAM].[dbo].[HCRDiabetesClinical]
WHERE InTestWindowFLG = 'N'
"

df <- selectData(connection.string, query)
head(df)

set.seed(42)
```

```
p <- SupervisedModelDevelopmentParams$new()
p$df <- df
p$type <- "classification"
p$impute <- TRUE
p$grainCol <- "PatientEncounterID"
p$predictedCol <- "ThirtyDayReadmitFLG"
p$debug <- FALSE
p$cores <- 1

# Run Lasso
lasso <- LassoDevelopment$new(p)
lasso$run()

set.seed(42)
# Run Random Forest
rf <- RandomForestDevelopment$new(p)
rf$run()

# Plot ROC
rocs <- list(rf$getROC(), lasso$getROC())
names <- c("Random Forest", "Lasso")
legendLoc <- "bottomright"
plotROCs(rocs, names, legendLoc)

# Plot PR Curve
rocs <- list(rf$getPRCurve(), lasso$getPRCurve())
names <- c("Random Forest", "Lasso")
legendLoc <- "bottomleft"
plotPRCurve(rocs, names, legendLoc)

print(proc.time() - ptm)
```

`LinearMixedModelDeployment` *Deploy a production-ready predictive Random Forest model*

Description

This step allows one to

- Create a final model on all of your training data
- Automatically save the model
- Run the model against test data to generate predictions
- Push these predictions to SQL Server

Usage

```
LinearMixedModelDeployment(type, df,
grainCol, personCol, testWindowCol, predictedCol, impute, debug)
```

Arguments

type	The type of model (either 'regression' or 'classification')
df	Dataframe whose columns are used for calc.
grainCol	Optional. The dataframe's column that has IDs pertaining to the grain. No ID columns are truly needed for this step.
personCol	The data frame's columns that represents the patient/person
testWindowCol	Y or N. This column dictates the split between model training and test sets. Those rows with N in this column indicate the training set while those that have Y indicate the test set
predictedCol	Column that you want to predict. If you're doing classification then this should be Y/N.
impute	For training df, set all-column imputation to F or T. This uses mean replacement for numeric columns and most frequent for factorized columns. F leads to removal of rows containing NULLs.
debug	Provides the user extended output to the console, in order to monitor the calculations throughout. Use T or F.

Format

An object of class `R6ClassGenerator` of length 24.

See Also

healthcareai

Examples

```
ptm <- proc.time()
library(healthcareai)

# setwd('C:/Yourscriptlocation/Useforwardslashes') # Uncomment if using csv

# Can delete this line in your work
csvfile <- system.file("extdata",
                       "HCRDiabetesClinical.csv",
                       package = "healthcareai")

# Replace csvfile with 'path/file'
df <- read.csv(file = csvfile,
               header = TRUE,
               na.strings = c("NULL", "NA", ""))
```

```
head(df)
str(df)

p <- SupervisedModelDeploymentParams$new()
p$type <- "classification"
p$df <- df
p$grainCol <- "PatientEncounterID"
p$personCol <- "PatientID"
p$testWindowCol <- "InTestWindowFLG"
p$predictedCol <- "ThirtyDayReadmitFLG"
p$impute <- TRUE
p$debug <- FALSE
p$useSavedModel <- FALSE
p$cores <- 1
p$writeToDB <- FALSE

dLMM <- LinearMixedModelDeployment$new(p)
dLMM$deploy()

df <- dLMM$getOutDf()
# Write to CSV (or JSON, MySQL, etc) using R syntax
# write.csv(df,'path/predictionsfile.csv')

print(proc.time() - ptm)



#### Classification example using diabetes data ####
# This example requires you to first create a table in SQL Server
# If you prefer to not use SAMD, execute this in SSMS to create output table:
# CREATE TABLE dbo.HCRDeployRegressionBASE(
# BindingID float, BindingNM varchar(255), LastLoadDTS datetime2,
# PatientEncounterID int, <--change to match inputID
# PredictedValueNBR decimal(38, 2),
# Factor1TXT varchar(255), Factor2TXT varchar(255), Factor3TXT varchar(255)
# )

# setwd('C:/Yourscriptlocation/Useforwardslashes') # Uncomment if using csv
ptm <- proc.time()
library(healthcareai)

connection.string <- "
driver={SQL Server};
server=localhost;
database=SAM;
trusted_connection=true
"

query <- "
SELECT
 [PatientEncounterID]
,[PatientID]           --Mixed model needs two ID columns
,[SystolicBPNBR]
```

```
,[LDLNBR]
,[A1CNBR]
,[GenderFLG]
,[ThirtyDayReadmitFLG]
,[InTestWindowFLG]
FROM [SAM].[dbo].[HCRDiabetesClinical]
--no WHERE clause, because we want train AND test
"

df <- selectData(connection.string, query)

head(df)
str(df)

p <- SupervisedModelDeploymentParams$new()
p$type <- "classification"
p$df <- df
p$grainCol <- "PatientEncounterID"
p$personCol <- "PatientID"
p$testWindowCol <- "InTestWindowFLG"
p$predictedCol <- "ThirtyDayReadmitFLG"
p$impute <- FALSE
p$debug <- FALSE
p$useSavedModel <- FALSE
p$cores <- 1
p$sqlConn <- connection.string
p$destSchemaTable <- "dbo.HCRDeployClassificationBASE"

lMM <- LinearMixedModelDeployment$new(p)
lMM$deploy()

print(proc.time() - ptm)




#### Regression example using diabetes data ####
# This example requires you to first create a table in SQL Server
# If you prefer to not use SAMD, execute this in SSMS to create output table:
# CREATE TABLE dbo.HCRDeployRegressionBASE(
#   BindingID float, BindingNM varchar(255), LastLoadDTS datetime2,
#   PatientEncounterID int, <--change to match inputID
#   PredictedValueNBR decimal(38, 2),
#   Factor1TXT varchar(255), Factor2TXT varchar(255), Factor3TXT varchar(255)
# )

# setwd('C:/Yourscriptlocation/Useforwardslashes') # Uncomment if using csv
ptm <- proc.time()
library(healthcareai)

connection.string <- "
driver={SQL Server};
server=localhost;
```

```
database=SAM;
trusted_connection=true
"

query <- "
SELECT
 [PatientEncounterID]
,[PatientID]           --Mixed model needs two ID columns
,[SystolicBPNBR]
,[LDLNBR]
,[A1CNBR]
,[GenderFLG]
,[ThirtyDayReadmitFLG]
,[InTestWindowFLG]
FROM [SAM].[dbo].[HCRDiabetesClinical]
--no WHERE clause, because we want train AND test
"

df <- selectData(connection.string, query)

head(df)
str(df)

p <- SupervisedModelDeploymentParams$new()
p$type <- "regression"
p$df <- df
p$grainCol <- "PatientEncounterID"
p$personCol <- "PatientID"
p$testWindowCol <- "InTestWindowFLG"
p$predictedCol <- "A1CNBR"
p$impute <- TRUE
p$debug <- FALSE
p$useSavedModel <- FALSE
p$cores <- 1
p$sqlConn <- connection.string
p$destSchemaTable <- "dbo.HCRDeployRegressionBASE"

lMM <- LinearMixedModelDeployment$new(p)
lMM$deploy()

print(proc.time() - ptm)
```

`LinearMixedModelDevelopment` *Compare predictive models, created on your data*

Description

This step allows one to create test models on your data and helps determine which performs best.

Usage

```
LinearMixedModelDevelopment(object, type, df,
grainCol, personCol, predictedCol, impute, debug)
```

Arguments

object	of SuperviseModelParameters class for $new() constructor
type	The type of model (either 'regression' or 'classification')
df	Dataframe whose columns are used for calc.
grainCol	The data frame's ID column pertaining to the grain
personCol	The data frame's ID column pertaining to the person/patient
predictedCol	Column that you want to predict. If you're doing classification then this should be Y/N.
impute	Set all-column imputation to F or T. This uses mean replacement for numeric columns and most frequent for factorized columns. F leads to removal of rows containing NULLs.
debug	Provides the user extended output to the console, in order to monitor the calculations throughout. Use T or F.

Format

An object of class `R6ClassGenerator` of length 24.

References

http://healthcare.ai/

See Also

healthcareai

Examples

```
### Built-in example; Doing classification
library(healthcareai)
library(lme4)

df <- sleepstudy

str(df)

# Create binary column for classification
df$ReactionFLG <- ifelse(df$Reaction > 300, "Y", "N")
df$Reaction <- NULL

set.seed(42)
p <- SupervisedModelDevelopmentParams$new()
```

```
p$df <- df
p$type <- "classification"
p$impute <- TRUE
p$personCol <- "Subject"  # Think of this as PatientID
p$predictedCol <- "ReactionFLG"
p$debug <- FALSE
p$cores <- 1

# Create Mixed Model
lmm <- LinearMixedModelDevelopment$new(p)
lmm$run()

### Doing regression
library(healthcareai)

# SQL query and connection goes here - see SelectData function.

df <- sleepstudy

# Add GrainID, which is equivalent to PatientEncounterID
df$GrainID <- seq.int(nrow(df))

str(df)

set.seed(42)
p <- SupervisedModelDevelopmentParams$new()
p$df <- df
p$type <- "regression"
p$impute <- TRUE
p$grainCol <- "GrainID"  # Think of this as PatientEnounterID
p$personCol <- "Subject"  # Think of this as PatientID
p$predictedCol <- "Reaction"
p$debug <- FALSE
p$cores <- 1

# Create Mixed Model
lmm <- LinearMixedModelDevelopment$new(p)
lmm$run()

#### Example using csv data ####
library(healthcareai)
# setwd('C:/Your/script/location') # Needed if using YOUR CSV file
ptm <- proc.time()

# Can delete this line in your work
csvfile <- system.file("extdata", "HCRDiabetesClinical.csv", package = "healthcareai")
#Replace csvfile with "path/to/yourfile"
df <- read.csv(file = csvfile, header = TRUE, na.strings = c("NULL", "NA", ""))

head(df)

df$InTestWindowFLG <- NULL
```

```
set.seed(42)

p <- SupervisedModelDevelopmentParams$new()
p$df <- df
p$type <- "classification"
p$impute <- TRUE
p$grainCol <- "PatientEncounterID"
p$personCol <- "PatientID"
p$predictedCol <- "ThirtyDayReadmitFLG"
p$debug <- FALSE
p$cores <- 1

# Create Mixed Model
lmm <- LinearMixedModelDevelopment$new(p)
lmm$run()

set.seed(42)
# Run Lasso
# Lasso <- LassoDevelopment$new(p)
# Lasso$run()
print(proc.time() - ptm)


#### This example is specific to Windows and is not tested.
#### Example using SQL Server data ####
# This example requires that you alter your connection string / query
# to read in your own data

ptm <- proc.time()
library(healthcareai)

connection.string <- "
driver={SQL Server};
server=localhost;
database=SAM;
trusted_connection=true
"

query <- "
SELECT
 [PatientEncounterID]
,[PatientID]
,[SystolicBPNBR]
,[LDLNBR]
,[A1CNBR]
,[GenderFLG]
,[ThirtyDayReadmitFLG]
,[InTestWindowFLG]
FROM [SAM].[dbo].[HCRDiabetesClinical]
--no WHERE clause, because we want train AND test
"

df <- selectData(connection.string, query)
```

```
head(df)

df$InTestWindowFLG <- NULL

set.seed(42)

p <- SupervisedModelDevelopmentParams$new()
p$df <- df
p$type <- "classification"
p$impute <- TRUE
p$grainCol <- "PatientEncounterID"
p$personCol <- "PatientID"
p$predictedCol <- "ThirtyDayReadmitFLG"
p$debug <- FALSE
p$cores <- 1

# Create Mixed Model
lmm <- LinearMixedModelDevelopment$new(p)
lmm$run()

# Remove person col, since RF can't use it
df$personCol <- NULL
p$df <- df
p$personCol <- NULL

set.seed(42)
# Run Random Forest
rf <- RandomForestDevelopment$new(p)
rf$run()

# Plot ROC
rocs <- list(lmm$getROC(), rf$getROC())
names <- c("Linear Mixed Model", "Random Forest")
legendLoc <- "bottomright"
plotROCs(rocs, names, legendLoc)

# Plot PR Curve
rocs <- list(lmm$getPRCurve(), rf$getPRCurve())
names <- c("Linear Mixed Model", "Random Forest")
legendLoc <- "bottomleft"
plotPRCurve(rocs, names, legendLoc)

print(proc.time() - ptm)
```

`orderByDate` *Order the rows in a data frame by date*

Description

Returns a data frame that's ordered by its date column

Usage

```
orderByDate(df, dateCol, descending = FALSE)
```

Arguments

df	A data frame
dateCol	Name of column in data frame that contains dates
descending	Boolean for whether the output should be in descending order

Value

A data frame ordered by date column

References

http://healthcare.ai

See Also

healthcareai

Examples

```
df <- data.frame(date=c('2009-01-01','2010-01-01','2009-03-08','2009-01-19'),
                 a=c(1,2,3,4))
dfResult <- orderByDate(df,'date', descending=FALSE)
head(dfResult)
```

---

percentDataAvailableInDateRange

Find the percent of a column that's filled

Description

Shows what percentage of data is avilable (potentially within a specified date range)

Usage

```
percentDataAvailableInDateRange(df, dateColumn = NULL,
  startInclusive = NULL, endExclusive = NULL)
```

Arguments

df	A dataframe
dateColumn	Optional string representing a date column of interest
startInclusive	Optional string in the in this date style: 'YYYY-MM-DD'
endExclusive	Optional string in the in this date style: 'YYYY-MM-DD'

Value

A labeled numeric vector, representing each column in input df

References

http://healthcare.ai

See Also

healthcareai

Examples

```
df <- data.frame(a = c(1,2,NA,NA),
                 b = c('m','f','m','f'),
                 c = c(0.7,NA,2.4,-4),
                 d = c(100,300,200,NA),
                 e = c(400,500,NA,504),
                 datecol = c('2012-01-01','2012-01-02',
                             '2012-01-03','2012-01-07'))

out <- percentDataAvailableInDateRange(df = df, # <- Only required argument
                                       dateColumn = 'datecol',
                                       startInclusive = '2012-01-01',
                                       endExclusive = '2012-01-08')
out
```

---

plotPRCurve *Plot PR Curves from SupervisedModel classes*

Description

Plot PRCurves calculated by children classes of SupervisedModel

Usage

```
plotPRCurve(PRCurves, names, legendLoc)
```

Arguments

PRCurves	A vector/array/list of PR curves
names	A vector of algorithm/class names
legendLoc	Location of the legend string to display

References

http://healthcare.ai

See Also

healthcareai

---

plotProfiler *Display availability feature profile over time*

Description

Shows what percentage of data is avilable after a particular starting time period.

Usage

```
plotProfiler(listOfVectors)
```

Arguments

listOfVectors	A list of vectors, where first vector has the hours and subsequent vectors represent the features and how much they're filled for each of the hours. Usually populated by featureAvailabilityProfiler()

Value

Nothing

References

http://healthcare.ai

See Also

healthcareai

Examples

```
lis <- list()
# Establish hour range/sequence
lis$hoursSinceAdmit <- c(0,1,3,6,12,24)

# Add features and their percent full for each hour
lis$BP <- c(40, 45, 65, 78, 80, 90)
lis$LDL <- c(10, 30, 40, 70, 100, 120)

plotProfiler(lis)
```

`plotROCs` *Plot ROCs from SupervisedModel classes*

Description

Plot ROCs calculated by children classes of SupervisedModel

Usage

```
plotROCs(rocs, names, legendLoc)
```

Arguments

`rocs`	A vector/array/list of ROC values
`names`	A vector of algorithm/class names
`legendLoc`	Location of the legend string to display

References

http://healthcare.ai

See Also

healthcareai

RandomForestDeployment *Deploy a production-ready predictive RandomForest model*

Description

This step allows one to

- Create a final model on all of your training data
- Automatically save the model
- Run the model against test data to generate predictions
- Push these predictions to SQL Server

Usage

```
RandomForestDeployment(type, df, grainCol, testWindowCol,
predictedCol, impute, debug)
```

Arguments

type	The type of model (either 'regression' or 'classification')
df	Dataframe whose columns are used for calc.
grainCol	The dataframe's column that has IDs pertaining to the grain
testWindowCol	Y or N. This column dictates the split between model training and test sets. Those rows with N in this column indicate the training set while those that have Y indicate the test set
predictedCol	Column that you want to predict. If you're doing classification then this should be Y/N.
impute	For training df, set all-column imputation to F or T. This uses mean replacement for numeric columns and most frequent for factorized columns. F leads to removal of rows containing NULLs.
debug	Provides the user extended output to the console, in order to monitor the calculations throughout. Use T or F.

Format

An object of class R6ClassGenerator of length 24.

See Also

healthcareai

Examples

```
#### Example using csv data ####
ptm <- proc.time()
library(healthcareai)

# setwd('C:/Yourscriptlocation/Useforwardslashes') # Uncomment if using csv

# Can delete this line in your work
csvfile <- system.file("extdata",
                       "HCRDiabetesClinical.csv",
                       package = "healthcareai")

# Replace csvfile with 'path/file'
df <- read.csv(file = csvfile,
               header = TRUE,
               na.strings = c("NULL", "NA", ""))

head(df)

# Remove unnecessary columns
df$PatientID <- NULL

p <- SupervisedModelDeploymentParams$new()
p$type <- "classification"
p$df <- df
p$grainCol <- "PatientEncounterID"
p$testWindowCol <- "InTestWindowFLG"
p$predictedCol <- "ThirtyDayReadmitFLG"
p$impute <- TRUE
p$debug <- FALSE
p$useSavedModel <- FALSE
p$cores <- 1
p$writeToDB <- FALSE

dL <- RandomForestDeployment$new(p)
dL$deploy()

df <- dL$getOutDf()
# Write to CSV (or JSON, MySQL, etc) using R syntax
# write.csv(df,'path/predictionsfile.csv')

print(proc.time() - ptm)



#### Classification example using SQL Server data ####
# If pushing predictions to SQL Server, first create a table
# If you prefer to not use SAMD, execute this in SSMS to create output table:
# CREATE TABLE dbo.HCRDeployClassificationBASE(
#   BindingID float, BindingNM varchar(255), LastLoadDTS datetime2,
#   PatientEncounterID int, <--change to match inputID
```

```
#   PredictedProbNBR decimal(38, 2),
#   Factor1TXT varchar(255), Factor2TXT varchar(255), Factor3TXT varchar(255)
# )
ptm <- proc.time()
library(healthcareai)

connection.string <- "
driver={SQL Server};
server=localhost;
database=SAM;
trusted_connection=true
"

query <- "
SELECT
 [PatientEncounterID] --Only need one ID column for random forest
,[SystolicBPNBR]
,[LDLNBR]
,[A1CNBR]
,[GenderFLG]
,[ThirtyDayReadmitFLG]
,[InTestWindowFLG]
FROM [SAM].[dbo].[HCRDiabetesClinical]
--no WHERE clause, because we want train AND test
"

df <- selectData(connection.string, query)

head(df)
str(df)

p <- SupervisedModelDeploymentParams$new()
p$type <- "classification"
p$df <- df
p$grainCol <- "PatientEncounterID"
p$testWindowCol <- "InTestWindowFLG"
p$predictedCol <- "ThirtyDayReadmitFLG"
p$impute <- TRUE
p$debug <- FALSE
p$useSavedModel <- FALSE
p$cores <- 1
p$sqlConn <- connection.string
p$destSchemaTable <- "dbo.HCRDeployClassificationBASE"

dL <- RandomForestDeployment$new(p)
dL$deploy()

print(proc.time() - ptm)



#### Regression example using SQL Server data ####
```

```
# If pushing predictions to SQL Server, first create a table
# If you prefer to not use SAMD, execute this in SSMS to create output table:
# CREATE TABLE dbo.HCRDeployRegressionBASE(
#   BindingID float, BindingNM varchar(255), LastLoadDTS datetime2,
#   PatientEncounterID int, <--change to match inputID
#   PredictedValueNBR decimal(38, 2),
#   Factor1TXT varchar(255), Factor2TXT varchar(255), Factor3TXT varchar(255)
# )
ptm <- proc.time()
library(healthcareai)

connection.string <- "
driver={SQL Server};
server=localhost;
database=SAM;
trusted_connection=true
"

query <- "
SELECT
 [PatientEncounterID] --Only need one ID column for random forest
,[SystolicBPNBR]
,[LDLNBR]
,[A1CNBR]
,[GenderFLG]
,[ThirtyDayReadmitFLG]
,[InTestWindowFLG]
FROM [SAM].[dbo].[HCRDiabetesClinical]
--no WHERE clause, because we want train AND test
"

df <- selectData(connection.string, query)

head(df)
str(df)

p <- SupervisedModelDeploymentParams$new()
p$type <- "regression"
p$df <- df
p$grainCol <- "PatientEncounterID"
p$testWindowCol <- "InTestWindowFLG"
p$predictedCol <- "A1CNBR"
p$impute <- TRUE
p$debug <- FALSE
p$useSavedModel <- FALSE
p$cores <- 1
p$sqlConn <- connection.string
p$destSchemaTable <- "dbo.HCRDeployRegressionBASE"

dL <- RandomForestDeployment$new(p)
dL$deploy()

print(proc.time() - ptm)
```

RandomForestDevelopment

Compare predictive models, created on your data

Description

This step allows you to create a random forest model, based on your data.

Usage

```
RandomForestDevelopment(object, type, df, grainCol, predictedCol,
impute, debug)
```

Arguments

object	of SuperviseModelParameters class for $new() constructor
type	The type of model (either 'regression' or 'classification')
df	Dataframe whose columns are used for calc.
grainCol	Optional. The dataframe's column that has IDs pertaining to the grain. No ID columns are truly needed for this step.
predictedCol	Column that you want to predict. If you're doing classification then this should be Y/N.
impute	Set all-column imputation to F or T. This uses mean replacement for numeric columns and most frequent for factorized columns. F leads to removal of rows containing NULLs.
debug	Provides the user extended output to the console, in order to monitor the calculations throughout. Use T or F.

Format

An object of class R6ClassGenerator of length 24.

References

http://hctools.org/

See Also

LassoDevelopment

LinearMixedModelDevelopment

healthcareai

Examples

```
#### Example using iris dataset ####
ptm <- proc.time()
library(healthcareai)

data(iris)
head(iris)

set.seed(42)

p <- SupervisedModelDevelopmentParams$new()
p$df <- iris
p$type <- "regression"
p$impute <- TRUE
p$grainCol <- ""
p$predictedCol <- "Sepal.Width"
p$debug <- FALSE
p$cores <- 1

# Run Lasso
lasso <- LassoDevelopment$new(p)
lasso$run()

set.seed(42)
# Run RandomForest
rf <- RandomForestDevelopment$new(p)
rf$run()

print(proc.time() - ptm)

#### Example using csv data ####
library(healthcareai)
# setwd('C:/Your/script/location') # Needed if using YOUR CSV file
ptm <- proc.time()

# Can delete this line in your work
csvfile <- system.file("extdata",
                       "HCRDiabetesClinical.csv",
                       package = "healthcareai")

# Replace csvfile with 'your/path'
df <- read.csv(file = csvfile,
               header = TRUE,
               na.strings = c("NULL", "NA", ""))

head(df)

df$PatientID <- NULL
df$InTestWindowFLG <- NULL

set.seed(42)
```

```
p <- SupervisedModelDevelopmentParams$new()
p$df <- df
p$type <- "regression"
p$impute <- TRUE
p$grainCol <- "PatientEncounterID"
p$predictedCol <- "A1CNBR"
p$debug <- FALSE
p$cores <- 1

# Run Lasso
lasso <- LassoDevelopment$new(p)
lasso$run()

set.seed(42)
# Run Random Forest
rf <- RandomForestDevelopment$new(p)
rf$run()

print(proc.time() - ptm)


#### Example using SQL Server data ####

ptm <- proc.time()
library(healthcareai)

connection.string <- "
driver={SQL Server};
server=localhost;
database=SAM;
trusted_connection=true
"

query <- "
SELECT
[PatientEncounterID]
,[SystolicBPNBR]
,[LDLNBR]
,[A1CNBR]
,[GenderFLG]
,[ThirtyDayReadmitFLG]
,[InTestWindowFLG]
FROM [SAM].[dbo].[HCRDiabetesClinical]
WHERE InTestWindowFLG = 'N'
"

df <- selectData(connection.string, query)
head(df)

df$InTestWindowFLG <- NULL

set.seed(42)
```

```
p <- SupervisedModelDevelopmentParams$new()
p$df <- df
p$type <- "classification"
p$impute <- TRUE
p$grainCol <- "PatientEncounterID"
p$predictedCol <- "ThirtyDayReadmitFLG"
p$debug <- FALSE
p$cores <- 1

# Run Lasso
lasso <- LassoDevelopment$new(p)
lasso$run()

set.seed(42)
# Run Random Forest
rf <- RandomForestDevelopment$new(p)
rf$run()

# Plot ROC
rocs <- list(rf$getROC(), lasso$getROC())
names <- c("Random Forest", "Lasso")
legendLoc <- "bottomright"
plotROCs(rocs, names, legendLoc)

# Plot PR Curve
rocs <- list(rf$getPRCurve(), lasso$getPRCurve())
names <- c("Random Forest", "Lasso")
legendLoc <- "bottomleft"
plotPRCurve(rocs, names, legendLoc)

print(proc.time() - ptm)
```

`removeColsWithAllSameValue` *Remove columns from a data frame when those columns have the same values in each row*

Description

Remove columns from a data frame when all of their rows are the same value (after removing NA's)

Usage

```
removeColsWithAllSameValue(df)
```

Arguments

`df`	A data frame

Value

A data frame with zero-variance columns removed

References

http://healthcare.ai

See Also

healthcareai

Examples

```
df <- data.frame(a=c(1,1,1),
                 b=c('a','b','b'),
                 c=c('a','a','a'),
                 d=c(NA,'1',NA))
dfResult <- removeColsWithAllSameValue(df)
head(dfResult)
```

---

removeRowsWithNAInSpecCol

Remove rows where specified col is NA

Description

Remove rows from a data frame where a particular col is NA

Usage

```
removeRowsWithNAInSpecCol(df, desiredCol)
```

Arguments

df	A data frame to be altered
desiredCol	A column name in the df (in ticks)

Value

dfResult The input data frame with rows removed

References

http://healthcare.ai

See Also

healthcareai

Examples

```
df <- data.frame(a=c(1,2,3),b=c('Y','N',NA),c=c(NA,'Y','N'))
dfResult <- removeRowsWithNAInSpecCol(df,'b')
head(dfResult)
```

`returnColsWithMoreThanFiftyCategories` *Return vector of columns in a data frame with greater than 50 categories*

Description

Returns a vector of the names of the columns that have more than 50 categories

Usage

```
returnColsWithMoreThanFiftyCategories(df)
```

Arguments

`df`	A data frame

Value

A vector that contains the names of the columns with greater than 50 categories

References

http://healthcare.ai

See Also

healthcareai

Examples

```
df <- data.frame(a=c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                     1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
                   b=c('a','b','c','d','e','f','g','h','i','j','k','l','m','n',
                       'o','p','q','r','s','t','u','v','w','x','y','z','aa',
                       'bb','cc','dd','ee','ff','gg','hh','ii','jj','kk','ll',
                       'mm','nn','oo','pp','qq','rr','ss','tt','uu','vv','ww',
                       'xx','yy'))
colList <- returnColsWithMoreThanFiftyCategories(df)
colList
```

`RiskAdjustedComparisons` *Make risk adjusted comparisons between groups/units or years/months*

Description

This class allows you to create a model based on the performance of many groups in a cohort (besides group A, for example) and see how well group A does against what the model would predict. Ranking each of the groups this way provides a sense of which group's doing best in terms of a particular measure.

Usage

```
RiskAdjustedComparisons(df, predictedCol, groupCol, impute)
```

Arguments

`df`	Dataframe whose columns are used for calc.
`predictedCol`	Column that you want to predict.
`groupCol`	Column that we'll use to differentiate
`impute`	Set all-column imputation to F or T.

Format

An object of class `R6ClassGenerator` of length 24.

References

http://healthcare.ai

See Also

healthcareai

Examples

```
#### Example using SQL data ####


library(healthcareai)

connection.string <- "
driver={SQL Server};
server=localhost;
database=SAM;
trusted_connection=true
```

```
"

query <- "
SELECT
[PatientEncounterID]
,[PatientID]
,[SystolicBPNBR]
,[LDLNBR]
,[A1CNBR]
,[GenderFLG]
,[ThirtyDayReadmitFLG]
,[InTestWindowFLG]
FROM [SAM].[dbo].[HCRDiabetesClinical]
"

df <- selectData(connection.string, query)

p <- SupervisedModelDevelopmentParams$new()
p$df <- df
p$groupCol <- "GenderFLG"
p$impute <- TRUE
p$predictedCol <- "ThirtyDayReadmitFLG"
p$debug <- FALSE
p$cores <- 1

riskAdjComp <- RiskAdjustedComparisons$new(p)
riskAdjComp$run()
```

`selectData` *Pull data into R via an ODBC connection*

Description

Select data from an ODBC database and return the results as a data frame.

Usage

```
selectData(MSSQLConnectionString = NULL, query, SQLiteFileName = NULL,
  randomize = FALSE, connectionString = NULL)
```

Arguments

`MSSQLConnectionString`	A string specifying the driver, server, database, and whether Windows Authentication will be used. Omit if using SQLite.
`query`	The SQL query (in ticks or quotes).
`SQLiteFileName`	A string. If your database type is SQLite, here one specifies the database file to query from.

randomize Boolean that dictates whether returned rows are randomized

connectionString
DEPRECATED. Use MSSQLConnectionString

Value

df A data frame containing the selected rows

References

http://healthcare.ai

See Also

healthcareai

Examples

```
# This example is specific to SQL Server

# To instead pull data from Oracle see here
# https://cran.r-project.org/web/packages/ROracle/ROracle.pdf
# To pull data from MySQL see here
# https://cran.r-project.org/web/packages/RMySQL/RMySQL.pdf
# To pull data from Postgres see here
# https://cran.r-project.org/web/packages/RPostgreSQL/RPostgreSQL.pdf

connectionString <- '
  driver={SQL Server};
  server=localhost;
  database=SAM;
  trustedConnection=true
  '

query <- '
  SELECT
    A1CNBR
  FROM SAM.dbo.HCRDiabetesClinical
  '

df <- selectData(connectionString, query)
head(df)


# SQLite example
query <- '
  SELECT *
  FROM HCRDiabetesClinical
  '
# Loads sample database; replace with your own SQLite db file
```

```
sqliteFile <- system.file("extdata",
                          "unit-test.sqlite",
                          package = "healthcareai")

df <- selectData(query = query,
                 SQLiteFileName = sqliteFile)
head(df)
```

`SupervisedModelDeployment`

Deploy predictive models, created on your data

Description

This step allows one to create deploy models on your data and helps determine which performs best.

Usage

```
SupervisedModelDeployment(object)
```

Arguments

`object`	of SupervisedModelDeploymentParams class for $new() constructor

Format

An object of class `R6ClassGenerator` of length 24.

References

http://healthcare.ai

See Also

`healthcareai`

SupervisedModelDeploymentParams *SupervisedModelDeploymentParams class to set up parameters required to build SupervisedModelDeployment class*

Description

This step allows one to create deploy models on your data and helps determine which performs best.

Usage

```
SupervisedModelDeploymentParams
```

Format

An object of class `R6ClassGenerator` of length 24.

References

http://healthcare.ai

See Also

healthcareai

SupervisedModelDevelopment *Compare predictive models, created on your data*

Description

This step allows one to create test models on your data and helps determine which performs best.

Usage

```
SupervisedModelDevelopment(object)
```

Arguments

`object`	of SuperviseModelParameters class for $new() constructor

Format

An object of class `R6ClassGenerator` of length 24.

References

http://healthcare.ai

See Also

healthcareai

`SupervisedModelDevelopmentParams` *SupervisedModelDevelopmentParams class to set up parameters required to build SupervisedModel classes*

Description

This step allows one to create test models on your data and helps determine which performs best.

Usage

```
SupervisedModelDevelopmentParams
```

Format

An object of class `R6ClassGenerator` of length 24.

References

http://healthcare.ai

See Also

healthcareai

`writeData` *Write data to database*

Description

Write data frame to database table via ODBC connection #' @param connectionString A string specifying the driver, server, database, and whether Windows Authentication will be used.

Usage

```
writeData(MSSQLConnectionString = NULL, df, SQLiteFileName = NULL,
  tableName)
```

Arguments

MSSQLConnectionString	A string specifying the driver, server, database, and whether Windows Authentication will be used.
df	Dataframe that hold the tabular data
SQLiteFileName	A string. If dbtype is SQLite, here one specifies the database file to query from
tableName	String. Name of the table that receives the new rows

Value

Nothing

References

http://healthcare.ai

See Also

healthcareai

Examples

```
# This example is specific to SQL Server.

# To instead pull data from Oracle see here
# https://cran.r-project.org/web/packages/ROracle/ROracle.pdf
# To pull data from MySQL see here
# https://cran.r-project.org/web/packages/RMySQL/RMySQL.pdf
# To pull data from Postgres see here
# https://cran.r-project.org/web/packages/RPostgreSQL/RPostgreSQL.pdf

# Before running this example, create the table in SQL Server via
# CREATE TABLE [dbo].[HCRWriteData](
# [a] [float] NULL,
# [b] [float] NULL,
# [c] [varchar](255) NULL)

connectionString <- '
  driver={SQL Server};
  server=localhost;
  database=SAM;
  trustedConnection=true
  '

df <- data.frame(a=c(1,2,3),
                 b=c(2,4,6),
                 c=c('one','two','three'))
```

```
writeData(MSSQLConnectionString = connectionString,
          df = df,
          tableName = 'HCRWriteData')
```

Index