Package 'healthcareai'

September 11, 2017

Type Package

Title Tools for Healthcare Machine Learning

Version 1.0.0

Date 2017-09-08

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, XGBoost, Kmeans clustering, 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, RSQLite, xgboost

RoxygenNote 6.0.1

Suggests testthat

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

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

NeedsCompilation no

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

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

Repository CRAN

Date/Publication 2017-09-11 17:17:00 UTC

R topics documented:

`addSAMUtilityCols` *Add SAM utility columns to table*

Description

When working in a Health Catalyst Source Area Mart (SAM), utility columns are added automatically when running a non-R binding

Usage

```
addSAMUtilityCols(df)
```

Arguments

`df`	A dataframe

Value

A dataframe with three additional columns

References

http://healthcareai-r.readthedocs.io

See Also

healthcareai

Examples

```
df <- data.frame(a = c(1,2,NA,NA),
                 b = c(100,300,200,150))
head(df)
df <- addSAMUtilityCols(df)
head(df)
```

---

assignClusterLabels *Assign labels to the kmeans confusion matrix*

Description

Finds the correct label for a the kmeans confusion matrix. This function assumes that the most populated cluster is the correct one for a given label.

Usage

```
assignClusterLabels(cm)
```

Arguments

cm	A square dataframe that holds a confusion matrix. Rownames must correspond to correct labels.

Value

A character vector of the label names.

References

http://healthcare.ai

See Also

healthcareai

Examples

```
data(iris)
head(iris)
kmeans.fit <- kmeans(iris[,1:4],3)
labs <- iris[,5]
cls <- kmeans.fit[["cluster"]]
cm <- calculateConfusion(labels = labs, clusters = cls)
assignClusterLabels(cm)
```

`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://healthcareai-r.readthedocs.io

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
```

`calculateConfusion` *Generate confusion matrix of percentages*

Description

Generate confusion matrix and convert from raw counts to percentage of each label

Usage

```
calculateConfusion(labels, clusters)
```

Arguments

`labels`	A vector with countable unique items, usually a factor variable in a data frame, labeling each observation.
`clusters`	A vector with countable unique items, usually is the clustering results returned by kmeans(), No NA's.

Value

A confusion matrix of percentages.

References

http://healthcare.ai

See Also

healthcareai

Examples

```
data(iris)
head(iris)
kmeans.fit <- kmeans(iris[,1:4],3)
labs <- iris[,5]
cls <- kmeans.fit[["cluster"]]
calculateConfusion(labels = labs, clusters = cls)
```

`calculateCOV` *Calculate coefficient of variation*

Description

Find coefficient of variation by dividing vector standard deviation by the mean.

Usage

```
calculateCOV(vector)
```

Arguments

`vector`	A vector of numbers

Value

A scalar

References

http://healthcareai-r.readthedocs.io https://en.wikipedia.org/wiki/Coefficient_of_variation

See Also

healthcareai, findVariation

Examples

```
df <- data.frame(a = c(1,2,NA,NA),
                 b = c(100,300,200,150))
res1 <- calculateCOV(df$a)
res1

res2 <- calculateCOV(df$b)
res2
```

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://healthcareai-r.readthedocs.io

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://healthcareai-r.readthedocs.io

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://healthcareai-r.readthedocs.io

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://healthcareai-r.readthedocs.io

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://healthcareai-r.readthedocs.io

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://healthcareai-r.readthedocs.io

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://healthcareai-r.readthedocs.io

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)
```

---

countMissingData *Function to find proportion of NAs in each column of a dataframe or matrix*

Description

Finds the proportion of NAs in each column of a dataframe or matrix. NA possibilities that are defined: NA, "NA", "NAs", "na", NaN, "NaN" , "?", "??", "nil", "NULL", " ", "", "999". User has ability to define their own NA values by using the userNAs parameter. User defined NAs will be added to the list of already defined NAs.

Usage

```
countMissingData(x, userNAs = NULL)
```

Arguments

x	A data frame or matrix
userNAs	A vector of user defined NA values.

Value

A numeric vector of the proportion of NAs in each column.

References

http://healthcareai-r.readthedocs.io

See Also

healthcareai

Examples

```
bob <- data.frame(d = c("NULL", NA, "empty", 5, "?", 'nil', "NaN", " ", 2),
                  y = c("??", ' ', "999", 999, "tom", "5", 7, 10, 2),
                  l = rep(NA, 9),
                  a = c("blank", 0, "na", "None", "none", 3, 10, 4, "what"),
                  n = c(10, 5, 8, 1, NA, "NULL", NaN, "Nas", 2),
                  new = c(1, 2, 3, 4, 5, "void", 7, 8, "what"))
countMissingData(bob)
countMissingData(bob, userNAs = c("void", "what"))
```

`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://healthcareai-r.readthedocs.io

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
```

createAllCombinations *Find all possible unique combinations*

Description

For a given vector of, find all possible combinations of the values. When calculating, if two groups contain the same values, they are counted as the same if they only differ in terms of ordering.

Usage

```
createAllCombinations(vector)
```

Arguments

vector	A vector of strings or numbers.

Value

A list of sub-lists. Each sub-list represents one possible combination.

References

http://healthcareai-r.readthedocs.io

See Also

healthcareai findVariation createVarianceTallTable

Examples

```
vector <- c("LactateOrderHospital",
                  "LactateOrderProvSpecialtyDSC",
                  "LactateOrderProvNM")
res <- createAllCombinations(vector)

# Let's look at one possible combination
unlist(res[3])

# Look at all possible combinations
res
```

createVarianceTallTable *Transform a dataframe to be three columns and tall instead of wide*

Description

When dealing with a table that could be unexpectedly wide, it helps to instead fix its width and let it get tall (which makes it easy to insert into a pre-existing table). This assists the findVariation function.

Usage

```
createVarianceTallTable(df, categoricalCols, measure)
```

Arguments

df	A dataframe
categoricalCols	Vector of strings, representing categorical column(s)
measure	String, representing measure column

Value

A dataframe of eight columns. MeasureVolumeRaw denotes number of rows in the particular subgroup; MeasureVolumePercent denotes percent of rows in that subgroup as a percentage of the above subgroup (i.e., F within Gender); MeasureImpact is the subgroup COV * VolRaw (i.e., num of rows).

References

http://healthcareai-r.readthedocs.io

See Also

healthcareai findVariation

Examples

```
df <- data.frame(LactateOrderProvSpecDSC = c("Pulmonary Disease",
                                             "Family Medicine"),
                 LactateOrderProvNM = c("Hector Salamanca",
                                        "Gus Fring"),
                 COV = c(0.43,0.35),
                 VolumeRaw = c(2,3),
                 VolumePercent = c(0.32,0.78),
                 Impact = c(0.46,1.05),
                 AboveMeanCOVFLG = c('Y','N'),
                 AboveMeanVolumeFLG = c('N','Y'))
```

```
head(df)

categoricalCols <- c("LactateOrderProvSpecDSC","LactateOrderProvNM")

dfRes <- createVarianceTallTable(df = df,
                                 categoricalCols = categoricalCols,
                                 measure = "LOS")
head(dfRes)
```

---

dataScale *Center and scale columns in a numeric data frame*

Description

center and scale columns in a numeric data frame using means and standard deviations

Usage

```
dataScale(df)
```

Arguments

df	A numeric data frame

Value

A list that contains a vector of column means, a vector of column sds, and a scaled data frame.

References

http://healthcare.ai

See Also

healthcareai

Examples

```
df <- data.frame(a = c(2,1,3,2,4),b = c(NA,8,6,7,9))
res <- dataScale(df)
res
```

`distancePointLine` *Compute the distance of a point from a line*

Description

compute the distance of a point from a line given the x, y axes of the point and the slope and intercept of the line.

Usage

```
distancePointLine(x, y, slope, intercept)
```

Arguments

`x`	A numeric vector, x axis of the points
`y`	A numeric vector, y axis of the points
`slope`	A number, the slope of the line
`intercept`	A number, the intercept of the line

Value

A numeric vector, the length of the pointS from the line.

References

http://healthcare.ai

This helper funtion is originally found here http://paulbourke.net/geometry/pointlineplane/pointline.r

https://github.com/bryanhanson/ChemoSpecMarkeR/blob/master/R/findElbow.R

See Also

healthcareai

Examples

```
distancePointLine(c(2,0),c(3,2),-0.5,2.5)
```

`distancePointSegment` *Compute the distance of a point from a line segment*

Description

compute the distance of a point from a line segment given the x, y axes of the points.

Usage

```
distancePointSegment(px, py, x1, y1, x2, y2)
```

Arguments

`px`	A numeric vector, x axis of the points
`py`	A numeric vector, y axis of the points
`x1`	A number, x axis of the end point of the line segment
`y1`	A number, y axis of the end point of the line segment
`x2`	A number, x axis of the other end point of the line segment
`y2`	A number, y axis of the other end point of the line segment

Value

numeric vector, the length between the pointS and the line segment. If the intersection point is outside the line segment, return the distance to the nearest endpoint.

References

http://healthcare.ai

The helper funtion is originally found here http://paulbourke.net/geometry/pointlineplane/pointline.r

https://github.com/bryanhanson/ChemoSpecMarkeR/blob/master/R/findElbow.R

See Also

healthcareai

Examples

```
distancePointSegment(c(2,0),c(3,2),1,2,3,1)
```

`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://healthcareai-r.readthedocs.io

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://healthcareai-r.readthedocs.io

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
```

`findElbow` *Find the elbow in a curve*

Description

finds the elbow of a curve that is concave to the line connecting the first and last points.

Usage

```
findElbow(y)
```

Arguments

y	A numeric vector of more than 2 elements w/o NA's

Value

A number, the index of the elbow point.

References

http://healthcare.ai

This function is developed based on the following function in package ChemoSpecMarkeR https://github.com/bryanhanson/ChemoSpecMarkeR/blob/master/R/findElbow.R

The idea behind the function can be found here https://stackoverflow.com/questions/2018178/finding-the-best-trade-off-point-on-a-curve/2022348#2022348

See Also

healthcareai

Examples

```
y <- c(8.5, 4.9, 2.8, 2.5, 1.9, 1.1, 1.1, 0.9)
plot(y) # concave
findElbow(y)

y <- c(6,5.5,4,2,1.5)
plot(y) # not concave
## findElbow(y) will gave an error
```

---

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

Description

Search within subgroups and find trends that are six months of longer.

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://healthcareai-r.readthedocs.io

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')
```

---

findVariation *Find high variation*

Description

Search across subgroups and surface those that have coefficient of variation * volume above a particular threshold

Usage

```
findVariation(df, categoricalCols, measureColumn, dateCol = NULL,
  threshold = NULL)
```

Arguments

df	A dataframe
categoricalCols	Vector of strings representing categorical column(s)
measureColumn	Vector of strings representing measure column(s)
dateCol	Optional. A date(time) column to group by (done by month)
threshold	A scalar number, representing the minimum impact values that are returned

Value

A dataframe of eight columns. MeasureVolumeRaw denotes number of rows in the particular subgroup; MeasureVolumePercent denotes percent of rows in that subgroup as a percentage of the above subgroup (i.e., F within Gender); MeasureImpact is the subgroup COV * VolRaw (i.e., num of rows).

References

http://healthcareai-r.readthedocs.io

See Also

healthcareai calculateCOV createVarianceTallTable

Examples

```
df <- data.frame(Dept = c('A','A','A','B','B','B','B','B'),
                 Gender = c('F','M','M','M','M','F','F','F'),
                 LOS = c(3.2,NA,5,1.3,2.4,4,9,10))

categoricalCols <- c("Dept","Gender")

dfRes <- findVariation(df = df,
                       categoricalCols = categoricalCols,
                       measureColumn = "LOS")

dfRes
```

---

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://healthcareai-r.readthedocs.io

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://healthcareai-r.readthedocs.io

See Also

healthcareai

`getPipedValue` *Grab number after single pipe in pipe-delimited string*

Description

For a given string with a pipe, return the number that comes after the pipe

Usage

```
getPipedValue(string)
```

Arguments

`string`	A string with a pipe

Value

A number from the original string

See Also

healthcareai `findVariation` `createVarianceTallTable`

Examples

```
res <- getPipedValue('hello|23')
res
```

getPipedWordCount *Count number of words in pipe-delimited string*

Description

For a given string with pipe(s), count the number of word-like sections that are separated by pipes.

Usage

```
getPipedWordCount(string)
```

Arguments

string	A string with pipes

Value

A count of number of words in input string.

See Also

healthcareai findVariation createVarianceTallTable

Examples

```
res <- getPipedWordCount('hello|sir')
res
```

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://healthcareai-r.readthedocs.io

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 to 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 three-step process: First, loading, profiling, and feature engineering. Second, developing a model. Third, deploying and monitoring the model.

1. **Load and profile data**

   - Loading Data:
     - Use selectData to pull data directly from a SQL database
   - Profiling and Analyzing Data - One can get quite far in healthcare data analysis without ever going beyond this step:
     - featureAvailabilityProfiler will find how much data is present in each variable over time.

  - `countMissingData` finds the proportion of missing data in each variable.
  - `findVariation` and `variationAcrossGroups` are used to find variation across/between subgroups of data.
  - `findTrends` finds trends that are six months or longer.
  - `RiskAdjustedComparisons` compares groups in a risk adjusted fashion. See RiskAdjustement for a general introduction.
  - `calculateTargetedCorrelations` will calculate correlations for all numeric columns and a specified variable of interest.
  - `returnColsWithMoreThanFiftyCategories` shows which categorical columns have more than 50 categories.
  - `KmeansClustering` used to cluster data with or without an outcome variable
- Feature Engineering:
  - `convertDateTimeColToDummies` will convert a date variable into dummy columns of day, hour, etc. For seasonal pattern modeling.
  - `countDaysSinceFirstDate` shows days since first day in input column.
  - `groupedLOCF` carries last observed value forward. This is an imputation method for longitudinal data.

2. **Develop a machine learning model**

- Models:
  - `LassoDevelopment`: Used for regression or classification and does an especially good job with a lot of variables.
  - `RandomForestDevelopment`: Used for regression or classification and is well suited to non-linear data.
  - `XGBoostDevelopment`: Used for multi-class classification (problems where there are more than 2 classes). Well suited to non-linear data.
  - `LinearMixedModelDevelopment`: Best suited for longitudinal data and datasets with less than 100k rows and 50 variables. Can do classification or regression.
- Performance of Trained Models:
  - Area under the ROC curve or area under the precision-recall curve are used to evaluate the performance of classification models.
  - The mean squared error (MSE) and root mean squared error (RMSE) are used to evaluate the performance of regression problems.

Note: models are saved in the working directory after creation.

3. **Deploy and Monitor the Machine Learning Model**

- Deploy the Model:
  - Use `LassoDeployment`, `LinearMixedModelDeployment`, `RandomForestDeployment`, or `XGBoostDeployment` to load the model from development and predict against test data. The deployments can be tested locally, but eventually live on the production server.
  - Use `writeData` to push the predicted values into a SQL environment.
- Monitoring the model:
  - `generateAUC` is used to monitor performance over time. This should happen after the predictions can be validated with the result. If you're predicting 30-day readmissions, you can't validate until 30 days have passed since the predictions.

References

http://healthcareai-r.readthedocs.io

http://healthcare.ai

`ignoreSpecWarn` *Function to suppress specific warnings in unit tests*

Description

This function allows one to suppress one or more specific warnings. The intended purpose is to suppress warning messages that are expected when running specific unit tests, but are unrelated to the functionality being tested.

Usage

```
ignoreSpecWarn(code, wRegexps)
```

Arguments

`code`	A piece of code to run, for which the warnings should be suppressed
`wRegexps`	A vector of regular expressions corresponding to the warnings that should be suppressed

References

http://healthcareai-r.readthedocs.io

See Also

`healthcareai`

`imputeColumn` *Depreciated in favor of* `imputeDF`

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

`imputeDF` *Perform imputation on a dataframe*

Description

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

Usage

```
imputeDF(df, imputeVals = NULL)
```

Arguments

`df`	A dataframe of values with NAs.
`imputeVals`	A list of values to be used for imputation. If an unnamed list must be the same length as as the number of columns in ‘df‘ and the order of values in ‘imputeVals‘ should match the order of columns in ‘df‘. If named, names will be matched to names of ‘df‘, and you can provide a subset of columns in ‘df‘ (see @details).

Details

If ‘imputeVals‘ is a named list containing a subset of the columns in ‘df‘, columns that don’t have a value provided will have one calculated. If you wish to provide custom imputation values for a subset of columns and leave NAs in other columns, supply the value NA to ‘imputeVals‘ for those columns. #’

Value

A list. The first element, df, is the imputed dataframe. The second element, imputeVals, is a list of the imputation value used.

References

http://healthcareai-r.readthedocs.io

See Also

healthcareai

Examples

```
# Impute a single column
df <- data.frame(a=c(1,2,3,NA), b=c('Y','N','Y',NA),
   c=c(11,21,31,43), d=c('Y','N','N',NA))
df <- df['a'] # note df[,1] does not return a df!
out <- imputeDF(df)
dfOut <- out$df # imputed data frame
imputeVals <- out$imputeVals # imputed values
print(dfOut)
# Impute an entire data frame
df <- data.frame(a=c(1,2,3,NA), b=c('Y','N','Y',NA),
   c=c(11,21,31,43), d=c('Y','N','N',NA))
out <- imputeDF(df)
dfOut <- out$df # imputed data frame
imputeVals <- out$imputeVals # imputed values
print(dfOut)

# To impute using your own values (one per column)
df <- data.frame(a=c(1,2,3,NA), b=c('Y','N','Y',NA),
   c=c(11,21,31,43), d=c('Y','N','N',NA))
myValues <- list(2, 'Y', 26.5, 'N')
out <- imputeDF(df, myValues)
dfOut <- out$df # imputed data frame
print(dfOut)
```

---

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://healthcareai-r.readthedocs.io

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://healthcareai-r.readthedocs.io

See Also

healthcareai

Examples

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

isNumeric — *Check if a data frame only has numeric columns.*

Description

Check if a dataframe only has numeric columns

Usage

```
isNumeric(df)
```

Arguments

df	A dataframe

Value

A boolean

References

http://healthcare.ai

See Also

healthcareai

Examples

```
df <- data.frame(a=c(1,2,3),
                 b=c(NA,3,2))
isNumeric(df)
```

isTargetYN — *Tests whether predictedCol is Y/N. Allows for NAs to be present.*

Description

Returns a logical, TRUE or FALSE, depending on what is contained in the vector. If any NAs are present in the vector, they will be removed later on in development with removeRowsWithNAInSpecCol.

Usage

```
isTargetYN(x)
```

Arguments

x	A data frame column, matrix column, or vector

Value

A boolean

References

http://healthcareai-r.readthedocs.io

See Also

healthcareai

Examples

```
dat <- data.frame(a = c(0,1,1,0,0,0,1,1,1,0,1,0,1,0,1,0,1,1,1,0))
dat2 <- data.frame(a = c(3, 4, 5, 6, 7, 8, 9))
dat3 <- data.frame(a = c('Y', 'N', 'Y', 'N'))
dat4 <- data.frame(a = c('Y', 'N', 'Y', 'N', NA))
isTargetYN(dat[, 1])
isTargetYN(dat2[, 1])
isTargetYN(dat3[, 1])
isTargetYN(dat4[, 1])
```

KmeansClustering *Build clusters using kmeans()*

Description

This step allows you to use kmeans clustering to explore and group your data.

Usage

```
KmeansClustering(object, df, grainCol, labelCol, numOfClusters,
  usePCA, numOfPCA,impute, debug)
```

Arguments

object	of UnsupervisedModelParams class for $new() constructor
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. If left blank, row numbers are used for identification.

`labelCol`	Optional. Labels will not be used for clustering. Labels can be can be used for validation. The number of clusters should be the same as the number of labels. Functions getClusterLabels() and getConfusionMatrix() are only available if labelCol is provided. Generally, supervised models are a better choice if your goal is classification.
`numOfClusters`	Number of clusters you want to build. If left blank, will be determined automatically from the elbow plot.
`usePCA`	Optional. TRUE or FALSE. Default is FALSE. If TRUE, the method will use principle components as the new features to perform K-means clustering. This may accelerate convergence on high-dimension datasets.
`numOfPCA`	Optional. If using principle components, you may specify the number to use to perform K-means clustering. If left blank, it will be determined automatically from the scree (elbow) plot.
`impute`	Set all-column imputation to FALSE or TRUE. This uses mean replacement for numeric columns and most frequent for factorized columns. FALSE leads to removal of rows containing NULLs.
`debug`	Provides the user extended output to the console, in order to monitor the calculations throughout. Use TRUE or FALSE.

Format

An object of class `R6ClassGenerator` of length 24.

Details

This is an unsupervised method for clustering data. That is, no response variable is needed or used. If you want to examine how the data clusters by some labeled grouping, you can specify the grouping in `labelCol`, but the labels are not used in the clustering process. If you want to use labels to train the model see `LassoDevelopment` or `RandomForestDevelopment`.

Methods

The above describes params for initializing a new KmeansClustering class with `$new()`. Individual methods are documented below.

`$new()`

Initializes a new Kmeans Clustering class using the parameters saved in `p`, documented above. This method loads, cleans, and prepares data for clustering.
Usage: `$new(p)`

`$run()`

Calculates clusters, displays performance.
Usage:`$run()`

`$get2DClustersPlot()`

Displays the data and assigned clusters. PCA is used to visualize the top two priciple components for plotting. This is unrelated to variable reduction for clustering. Passing TRUE to this function will display grain IDs on the plot.
Usage: `$get2DClustersPlot()`

`$getOutDf()`

Returns the output dataframe for writing to SQL or CSV.
Usage: `$getOutDf()`

`$getConfusionMatrix()`

Returns a confusion matrix of assigned cluster vs. provided labels. Clusters are named based on maximum overlap with label. Only available if labelCol is specified. Rows are true labels, columns are assigned clusters.
Usage: `$getConfusionMatrix()`

`$getElbowPlot()`

Plots total within cluster error vs. number of clusters. Available if the number of clusters is unspecified.
Usage: `$getElbowPlot()`

`$getScreePlot()`

Plots total variance explained vs. number of principle components. Available if the number of principle components is unspecified.
Usage: `$getScreePlot()`

`$getKmeansFit()`

Returns all attributes of the kmeans fit object.
Usage: `$getKmeansFit()`

References

http://hctools.org/

https://github.com/bryanhanson/ChemoSpecMarkeR/blob/master/R/findElbow.R

See Also

healthcareai

Examples

```
#### Example using Diabetes dataset ####
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

set.seed(42)
p <- UnsupervisedModelParams$new()
p$df <- df
p$impute <- TRUE
p$grainCol <- "PatientEncounterID"
p$debug <- FALSE
p$cores <- 1
p$numOfClusters <- 3

# Run k means clustering
cl <- KmeansClustering$new(p)
cl$run()

# Get the 2D representation of the cluster solution
cl$get2DClustersPlot()

# Get the output data frame
dfOut <- cl$getOutDf()
head(dfOut)

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

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

data(iris)
head(iris)

set.seed(2017)

p <- UnsupervisedModelParams$new()
p$df <- iris
p$labelCol <- 'Species'
```

```
p$impute <- TRUE
p$debug <- FALSE
p$cores <- 1

# Run k means clustering
cl <- KmeansClustering$new(p)
cl$run()

# Get the 2D representation of the cluster solution
cl$get2DClustersPlot()

# Get the output data frame
dfOut <- cl$getOutDf()
head(dfOut)

## Write to CSV (or JSON, MySQL, etc) using plain R syntax
## write.csv(dfOut,'path/clusteringresult.csv')

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

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

Description

This step allows one to

- Load a saved model from `LassoDevelopment`
- Run the model against test data to generate predictions
- Push these predictions to SQL Server

Usage

```
LassoDeployment(type, df, grainCol, predictedCol, impute, debug, cores, modelName)
```

Arguments

`type`	The type of model (either 'regression' or 'classification')
`df`	Dataframe whose columns are used for new predictions. Data structure should match development as much as possible. Number of columns, names, types, grain, and predicted must be the same.
`grainCol`	The dataframe's column that has IDs pertaining to the grain
`predictedCol`	Column that you want to predict.
`impute`	For training df, set all-column imputation to T or F. If T, this uses values calculated in development. F leads to removal of rows containing NULLs and is not recommended.

debug	Provides the user extended output to the console, in order to monitor the calculations throughout. Use T or F.
cores	Number of cores you'd like to use. Defaults to 2.
modelName	Optional string. Can specify the model name. If used, you must load the same one in the deploy step.

Format

An object of class R6ClassGenerator of length 24.

Methods

The above describes params for initializing a new lassoDeployment class with $new(). Individual methods are documented below.

$new()

Initializes a new lasso deployment class using the parameters saved in p, documented above. This method loads, cleans, and prepares data for generating predictions.
Usage: $new(p)

$deploy()

Generate new predictions, calculate top factors, and prepare the output dataframe.
Usage:$deploy()

$getTopFactors()

Return the grain, all top factors, and their weights.
Usage: $getTopFactors(numberOfFactors = NA, includeWeights = FALSE)
Params:
- numberOfFactors: returns the top n factors. Defaults to all factors.
- includeWeights: If TRUE, returns weights associated with each factor.

$getOutDf()

Returns the output dataframe.
Usage: $getOutDf()

See Also

healthcareai

writeData

selectData

Examples

```
#### Classification Example using csv data ####
## 1. Loading data and packages.
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", ""))

df$PatientID <- NULL # Only one ID column (ie, PatientEncounterID) is needed remove this column

# Save a dataframe for validation later on
dfDeploy <- df[951:1000,]

## 2. Train and save the model using DEVELOP
print('Historical, development data:')
str(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()

## 3. Load saved model and use DEPLOY to generate predictions.
print('Fake production data:')
str(dfDeploy)

p2 <- SupervisedModelDeploymentParams$new()
p2$type <- "classification"
p2$df <- dfDeploy
p2$grainCol <- "PatientEncounterID"
p2$predictedCol <- "ThirtyDayReadmitFLG"
p2$impute <- TRUE
p2$debug <- FALSE
```

```
p2$cores <- 1

dL <- LassoDeployment$new(p2)
dL$deploy()

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

## 1. Loading data and packages.
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 lasso
,[SystolicBPNBR]
,[LDLNBR]
,[A1CNBR]
,[GenderFLG]
,[ThirtyDayReadmitFLG]
FROM [SAM].[dbo].[HCRDiabetesClinical]
"

df <- selectData(connection.string, query)

# Save a dataframe for validation later on
dfDeploy <- df[951:1000,]

## 2. Train and save the model using DEVELOP
print('Historical, development data:')
str(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()

## 3. Load saved model and use DEPLOY to generate predictions.
print('Fake production data:')
str(dfDeploy)

p2 <- SupervisedModelDeploymentParams$new()
p2$type <- "classification"
p2$df <- dfDeploy
p2$grainCol <- "PatientEncounterID"
p2$predictedCol <- "ThirtyDayReadmitFLG"
p2$impute <- TRUE
p2$debug <- FALSE
p2$cores <- 1

dL <- LassoDeployment$new(p2)
dL$deploy()
dfOut <- dL$getOutDf()

writeData(MSSQLConnectionString = connection.string,
          df = dfOut,
          tableName = 'HCRDeployClassificationBASE')

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)
# )

## 1. Loading data and packages.
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 lasso
,[SystolicBPNBR]
,[LDLNBR]
,[A1CNBR]
,[GenderFLG]
,[ThirtyDayReadmitFLG]
FROM [SAM].[dbo].[HCRDiabetesClinical]
"

df <- selectData(connection.string, query)

# Save a dataframe for validation later on
dfDeploy <- df[951:1000,]

## 2. Train and save the model using DEVELOP
print('Historical, development data:')
str(df)

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()

## 3. Load saved model and use DEPLOY to generate predictions.
print('Fake production data:')
str(dfDeploy)

p2 <- SupervisedModelDeploymentParams$new()
p2$type <- "regression"
p2$df <- dfDeploy
p2$grainCol <- "PatientEncounterID"
p2$predictedCol <- "A1CNBR"
p2$impute <- TRUE
p2$debug <- FALSE
```

```
p2$cores <- 1

dL <- LassoDeployment$new(p2)
dL$deploy()
dfOut <- dL$getOutDf()

writeData(MSSQLConnectionString = connection.string,
          df = dfOut,
          tableName = 'HCRDeployRegressionBASE')

print(proc.time() - ptm)


#### Classification example pulling from CSV and writing to SQLite ####

## 1. Loading data and packages.
ptm <- proc.time()
library(healthcareai)

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

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

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

df$PatientID <- NULL # Only one ID column (ie, PatientEncounterID) is needed

# Save a dataframe for validation later on
dfDeploy <- df[951:1000,]

## 2. Train and save the model using DEVELOP
print('Historical, development data:')
str(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()

## 3. Load saved model and use DEPLOY to generate predictions.
print('Fake production data:')
str(dfDeploy)

p2 <- SupervisedModelDeploymentParams$new()
p2$type <- "classification"
p2$df <- dfDeploy
p2$grainCol <- "PatientEncounterID"
p2$predictedCol <- "ThirtyDayReadmitFLG"
p2$impute <- TRUE
p2$debug <- FALSE
p2$cores <- 1

dL <- LassoDeployment$new(p2)
dL$deploy()
dfOut <- dL$getOutDf()

writeData(SQLiteFileName = sqliteFile,
          df = dfOut,
          tableName = 'HCRDeployClassificationBASE')

print(proc.time() - ptm)

#### Regression example pulling from CSV and writing to SQLite ####

## 1. Loading data and packages.
ptm <- proc.time()
library(healthcareai)

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

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

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

df$PatientID <- NULL # Only one ID column (ie, PatientEncounterID) is needed remove this column

# Save a dataframe for validation later on
dfDeploy <- df[951:1000,]

## 2. Train and save the model using DEVELOP
print('Historical, development data:')
```

```
str(df)

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()

## 3. Load saved model and use DEPLOY to generate predictions.
print('Fake production data:')
str(dfDeploy)

p2 <- SupervisedModelDeploymentParams$new()
p2$type <- "regression"
p2$df <- dfDeploy
p2$grainCol <- "PatientEncounterID"
p2$predictedCol <- "A1CNBR"
p2$impute <- TRUE
p2$debug <- FALSE
p2$cores <- 1

dL <- LassoDeployment$new(p2)
dL$deploy()
dfOut <- dL$getOutDf()

writeData(SQLiteFileName = sqliteFile,
          df = dfOut,
          tableName = 'HCRDeployRegressionBASE')

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. Lasso is a linear model, best suited for linearly separable data. It's fast to train and often a good starting point.

Usage

```
LassoDevelopment(type, df, grainCol, predictedCol, impute,
debug, cores, modelName)
```

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.
predictedCol	Column that you want to predict. If you're doing classification then this should be Y/N.
impute	Set all-column imputation to T or F. If T, this uses mean replacement for numeric columns and most frequent for factorized columns. F leads to removal of rows containing NULLs. Values are saved for deployment.
debug	Provides the user extended output to the console, in order to monitor the calculations throughout. Use T or F.
cores	Number of cores you'd like to use. Defaults to 2.
modelName	Optional string. Can specify the model name. If used, you must load the same one in the deploy step.

Format

An object of class `R6ClassGenerator` of length 24.

Methods

The above describes params for initializing a new lassoDevelopment class with `$new()`. Individual methods are documented below.

`$new()`

Initializes a new lasso development class using the parameters saved in `p`, documented above. This method loads, cleans, and prepares data for model training.
Usage: `$new(p)`

`$run()`

Trains model, displays feature importance and performance.
Usage:`$new()`

`$getPredictions()`

Returns the predictions from test data.
Usage: `$getPredictions()`

`$getROC()`

Returns the ROC curve object for `plotROCs`. Classification models only.
Usage: `$getROC()`

`$getPRCurve()`

Returns the PR curve object for `plotPRCurve`. Classification models only.
Usage: `$getROC()`

`$getAUROC()`

Returns the area under the ROC curve from testing for classification models.
Usage: `$getAUROC()`

`$getRMSE()`

Returns the RMSE from test data for regression models.
Usage: `$getRMSE()`

`$getMAE()`

Returns the RMSE from test data for regression models.
Usage: `$getMAE()`

References

http://healthcareai-r.readthedocs.io

See Also

RandomForestDevelopment

LinearMixedModelDevelopment

selectData

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()

cat(proc.time() - ptm,"\n")

#### 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

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()

cat(proc.time() - ptm,"\n")
```

```
#### 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
"
# This query should pull only rows for training. They must have a label.
query <- "
SELECT
[PatientEncounterID]
,[SystolicBPNBR]
,[LDLNBR]
,[A1CNBR]
,[GenderFLG]
,[ThirtyDayReadmitFLG]
FROM [SAM].[dbo].[HCRDiabetesClinical]
"
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)

cat(proc.time() - ptm,"\n")
```

`LinearMixedModelDeployment` *Deploy a production-ready predictive Linear Mixed Model model*

Description

This step allows one to

- Load a saved model from `LinearMixedModelDevelopment`
- Run the model against test data to generate predictions
- Push these predictions to SQL Server

The linear mixed model functionality works best with smaller data sets.

Usage

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

Arguments

`type`	The type of model (either 'regression' or 'classification')
`df`	Dataframe whose columns are used for new predictions. Data structure should match development as much as possible. Number of columns, names, types, grain, and predicted must be the same.
`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
`predictedCol`	Column that you want to predict.
`impute`	For training df, set all-column imputation to T or F. If T, this uses values calculated in development. F leads to removal of rows containing NULLs and is not recommended.
`debug`	Provides the user extended output to the console, in order to monitor the calculations throughout. Use T or F.
`cores`	Number of cores you'd like to use. Defaults to 2.
`modelName`	Optional string. Can specify the model name. If used, you must load the same one in the deploy step.

Format

An object of class `R6ClassGenerator` of length 24.

Methods

The above describes params for initializing a new linearMixedModelDeployment class with `$new()`. Individual methods are documented below.

`$new()`

Initializes a new linear mixed model deployment class using the parameters saved in `p`, documented above. This method loads, cleans, and prepares data for generating predictions.
Usage: `$new(p)`

`$deploy()`

Generate new predictions, calculate top factors, and prepare the output dataframe.
Usage:`$deploy()`

`$getTopFactors()`

Return the grain, all top factors, and their weights.
Usage: `$getTopFactors(numberOfFactors = NA, includeWeights = FALSE)`
Params:
- `numberOfFactors:` returns the top `n` factors. Defaults to all factors.
- `includeWeights:` If `TRUE`, returns weights associated with each factor.

`$getOutDf()`

Returns the output dataframe.
Usage: `$getOutDf()`

See Also

`healthcareai`

`writeData`

`selectData`

Examples

```
#### Classification Example using csv data ####
## 1. Loading data and packages.
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", ""))

# Save a dataframe for validation later on
dfDeploy <- df[951:1000,]

## 2. Train and save the model using DEVELOP
print('Historical, development data:')
str(df)

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

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

## 3. Load saved model and use DEPLOY to generate predictions.
print('Fake production data:')
str(dfDeploy)

p2 <- SupervisedModelDeploymentParams$new()
p2$type <- "classification"
p2$df <- dfDeploy
p2$grainCol <- "PatientEncounterID"
p2$personCol <- "PatientID"
p2$predictedCol <- "ThirtyDayReadmitFLG"
p2$impute <- TRUE
p2$debug <- FALSE
p2$cores <- 1

dL <- LinearMixedModelDeployment$new(p2)
dL$deploy()

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

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

```
## Not run:
#### 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)
# )

## 1. Loading data and packages.
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]
FROM [SAM].[dbo].[HCRDiabetesClinical]
"

df <- selectData(connection.string, query)

# Save a dataframe for validation later on
dfDeploy <- df[951:1000,]

## 2. Train and save the model using DEVELOP
print('Historical, development data:')
str(df)

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

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

## 3. Load saved model and use DEPLOY to generate predictions.
print('Fake production data:')
str(dfDeploy)

p2 <- SupervisedModelDeploymentParams$new()
p2$type <- "classification"
p2$df <- dfDeploy
p2$grainCol <- "PatientEncounterID"
p2$personCol <- "PatientID"
p2$predictedCol <- "ThirtyDayReadmitFLG"
p2$impute <- TRUE
p2$debug <- FALSE
p2$cores <- 1

dL <- LinearMixedModelDeployment$new(p2)
dL$deploy()
dfOut <- dL$getOutDf()

writeData(MSSQLConnectionString = connection.string,
          df = dfOut,
          tableName = 'HCRDeployClassificationBASE')

print(proc.time() - ptm)

## End(Not run)

## Not run:
#### 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)
# )

## 1. Loading data and packages.
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]
FROM [SAM].[dbo].[HCRDiabetesClinical]
"

df <- selectData(connection.string, query)

# Save a dataframe for validation later on
dfDeploy <- df[951:1000,]

## 2. Train and save the model using DEVELOP
print('Historical, development data:')
str(df)

set.seed(42)
p <- SupervisedModelDevelopmentParams$new()
p$df <- df
p$type <- "regression"
p$impute <- TRUE
p$grainCol <- "PatientEncounterID"
p$personCol <- "PatientID"
p$predictedCol <- "A1CNBR"
p$debug <- FALSE
p$cores <- 1

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

## 3. Load saved model and use DEPLOY to generate predictions.
dfDeploy$A1CNBR <- NULL # You won't know the response in production
print('Fake production data:')
str(dfDeploy)

p2 <- SupervisedModelDeploymentParams$new()
p2$type <- "regression"
p2$df <- dfDeploy
p2$grainCol <- "PatientEncounterID"
p2$personCol <- "PatientID"
p2$predictedCol <- "A1CNBR"
p2$impute <- TRUE
p2$debug <- FALSE
p2$cores <- 1

dL <- LinearMixedModelDeployment$new(p2)
```

```
dL$deploy()
dfOut <- dL$getOutDf()

writeData(MSSQLConnectionString = connection.string,
          df = dfOut,
          tableName = 'HCRDeployRegressionBASE')

print(proc.time() - ptm)

## End(Not run)

## Not run:
#### Classification example pulling from CSV and writing to SQLite ####


## 1. Loading data and packages.
ptm <- proc.time()
library(healthcareai)

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

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

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

# Save a dataframe for validation later on
dfDeploy <- df[951:1000,]

## 2. Train and save the model using DEVELOP
print('Historical, development data:')
str(df)

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

# Run Linear Mixed Model
lmm <- LinearMixedModelDevelopment$new(p)
```

```
lmm$run()

## 3. Load saved model and use DEPLOY to generate predictions.
print('Fake production data:')
str(dfDeploy)

p2 <- SupervisedModelDeploymentParams$new()
p2$type <- "classification"
p2$df <- dfDeploy
p2$grainCol <- "PatientEncounterID"
p2$personCol <- "PatientID"
p2$predictedCol <- "ThirtyDayReadmitFLG"
p2$impute <- TRUE
p2$debug <- FALSE
p2$cores <- 1

dL <- LinearMixedModelDeployment$new(p2)
dL$deploy()
dfOut <- dL$getOutDf()

writeData(SQLiteFileName = sqliteFile,
          df = dfOut,
          tableName = 'HCRDeployClassificationBASE')

print(proc.time() - ptm)

## End(Not run)

## Not run:
#### Regression example pulling from CSV and writing to SQLite ####

## 1. Loading data and packages.
ptm <- proc.time()
library(healthcareai)

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

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

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

# Save a dataframe for validation later on
dfDeploy <- df[951:1000,]

## 2. Train and save the model using DEVELOP
```

```
print('Historical, development data:')
str(df)

set.seed(42)
p <- SupervisedModelDevelopmentParams$new()
p$df <- df
p$type <- "regression"
p$impute <- TRUE
p$grainCol <- "PatientEncounterID"
p$personCol <- "PatientID"
p$predictedCol <- "A1CNBR"
p$debug <- FALSE
p$cores <- 1

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

## 3. Load saved model and use DEPLOY to generate predictions.
dfDeploy$A1CNBR <- NULL # You won't know the response in production
print('Fake production data:')
str(dfDeploy)

p2 <- SupervisedModelDeploymentParams$new()
p2$type <- "regression"
p2$df <- dfDeploy
p2$grainCol <- "PatientEncounterID"
p2$personCol <- "PatientID"
p2$predictedCol <- "A1CNBR"
p2$impute <- TRUE
p2$debug <- FALSE
p2$cores <- 1

dL <- LinearMixedModelDeployment$new(p2)
dL$deploy()
dfOut <- dL$getOutDf()

writeData(SQLiteFileName = sqliteFile,
          df = dfOut,
          tableName = 'HCRDeployRegressionBASE')

print(proc.time() - ptm)

## End(Not run)
```

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

Description

This step allows you to create a linear mixed model on your data. LMM is best suited to longitudinal data that has multiple entries for a each patient or physician. It will fit a slightly different linear model to each patient. This algorithm works best with linearly separable data sets. As data sets become longer than 100k rows or wider than 50 features, performance will suffer.

Usage

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

Arguments

`type`	The type of model (either 'regression' or 'classification')
`df`	Dataframe whose columns are used for calc.
`grainCol`	Optional. 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 T or F. If T, this uses mean replacement for numeric columns and most frequent for factorized columns. F leads to removal of rows containing NULLs. Values are saved for deployment.
`debug`	Provides the user extended output to the console, in order to monitor the calculations throughout. Use T or F.
`cores`	Number of cores you'd like to use. Defaults to 2.
`modelName`	Optional string. Can specify the model name. If used, you must load the same one in the deploy step.

Format

An object of class `R6ClassGenerator` of length 24.

Methods

The above describes params for initializing a new linearMixedModelDevelopment class with `$new()`. Individual methods are documented below.

`$new()`

Initializes a new linear mixed model development class using the parameters saved in `p`, documented above. This method loads, cleans, and prepares data for model training.
Usage: `$new(p)`

`$run()`

Trains model, displays feature importance and performance.
Usage:`$new()`

`$getPredictions()`

Returns the predictions from test data.
Usage: `$getPredictions()`

`$getROC()`

Returns the ROC curve object for `plotROCs`. Classification models only.
Usage: `$getROC()`

`$getPRCurve()`

Returns the PR curve object for `plotPRCurve`. Classification models only.
Usage: `$getROC()`

`$getAUROC()`

Returns the area under the ROC curve from testing for classification models.
Usage: `$getAUROC()`

`$getRMSE()`

Returns the RMSE from test data for regression models.
Usage: `$getRMSE()`

`$getMAE()`

Returns the RMSE from test data for regression models.
Usage: `$getMAE()`

References

http://healthcareai-r.readthedocs.io

See Also

`selectData`

`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)

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()
cat(proc.time() - ptm, '\n')

## Not run:
#### 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
"
```

```
# This query should pull only rows for training. They must have a label.
query <- "
SELECT
 [PatientEncounterID]
,[PatientID]
,[SystolicBPNBR]
,[LDLNBR]
,[A1CNBR]
,[GenderFLG]
,[ThirtyDayReadmitFLG]
FROM [SAM].[dbo].[HCRDiabetesClinical]
--no WHERE clause, because we want train AND test
"

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

set.seed(42)

p <- SupervisedModelDevelopmentParams$new()
p$df <- df
p$type <- "classification"
p$impute <- TRUE
p$grainCol <- "PatientEncounterID"
p$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)

cat(proc.time() - ptm, '\n')

## End(Not run)
```

`lineMagnitude` *Compute the distance between two points*

Description

compute the distance between two points given the x, y axes of the points.

Usage

```
lineMagnitude(x1, y1, x2, y2)
```

Arguments

x1	A numeric vector, x axis of the points
y1	A numeric vector, y axis of the points
x2	A numeric vector, x axis of the points
y2	A numeric vector, y axis of the points

Value

A numeric vector, the length between two points.

References

http://healthcare.ai

See Also

healthcareai

Examples

```
lineMagnitude(c(1,0),c(2,0),5,3)
```

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://healthcareai-r.readthedocs.io

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)
```

pcaAnalysis *Perform principle component analysis*

Description

performs PCA on numeric data frames

Usage

```
pcaAnalysis(df)
```

Arguments

df	A numeric data frame w/o NA's

Value

A list that contains the data frame of principle components and the proportion of variance explained by each PC.

References

http://healthcare.ai

See Also

healthcareai

Examples

```
data(iris)
head(iris)
df <- iris[,1:4]
res <- pcaAnalysis(df)
head(res[[1]])
```

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://healthcareai-r.readthedocs.io

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://healthcareai-r.readthedocs.io

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://healthcareai-r.readthedocs.io

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://healthcareai-r.readthedocs.io

See Also

healthcareai

RandomForestDeployment *Deploy a production-ready predictive RandomForest model*

Description

This step allows one to

- Load a saved model from RandomForestDevelopment
- Run the model against test data to generate predictions
- Push these predictions to SQL Server

Usage

```
RandomForestDeployment(type, df, grainCol, predictedCol, impute, debug, cores, modelName)
```

Arguments

type	The type of model (either 'regression' or 'classification')
df	Dataframe whose columns are used for new predictions. Data structure should match development as much as possible. Number of columns, names, types, grain, and predicted must be the same.
grainCol	The dataframe's column that has IDs pertaining to the grain
predictedCol	Column that you want to predict.
impute	For training df, set all-column imputation to T or F. If T, this uses values calculated in development. F leads to removal of rows containing NULLs and is not recommended.
debug	Provides the user extended output to the console, in order
cores	Number of cores you'd like to use. Defaults to 2.
modelName	Optional string. Can specify the model name. If used, you must load the same one in the deploy step.

Format

An object of class R6ClassGenerator of length 24.

Methods

The above describes params for initializing a new randomForestDeployment class with $new(). Individual methods are documented below.

$new()

Initializes a new random forest deployment class using the parameters saved in p, documented above. This method loads, cleans, and prepares data for generating predictions.
Usage: $new(p)

`$deploy()`

Generate new predictions, calculate top factors, and prepare the output dataframe.
Usage: `$deploy()`

`$getTopFactors()`

Return the grain, all top factors, and their weights.
Usage: `$getTopFactors(numberOfFactors = NA, includeWeights = FALSE)`
Params:
- `numberOfFactors:` returns the top n factors. Defaults to all factors.
- `includeWeights:` If `TRUE`, returns weights associated with each factor.

`$getOutDf()`

Returns the output dataframe.
Usage: `$getOutDf()`

See Also

healthcareai

writeData

selectData

Examples

```
#### Classification Example using csv data ####
## 1. Loading data and packages.
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", ""))

df$PatientID <- NULL # Only one ID column (ie, PatientEncounterID) is needed remove this column

# Save a dataframe for validation later on
dfDeploy <- df[951:1000,]

## 2. Train and save the model using DEVELOP
print('Historical, development data:')
```

```
str(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 RandomForest
RandomForest <- RandomForestDevelopment$new(p)
RandomForest$run()

## 3. Load saved model and use DEPLOY to generate predictions.
print('Fake production data:')
str(dfDeploy)

p2 <- SupervisedModelDeploymentParams$new()
p2$type <- "classification"
p2$df <- dfDeploy
p2$grainCol <- "PatientEncounterID"
p2$predictedCol <- "ThirtyDayReadmitFLG"
p2$impute <- TRUE
p2$debug <- FALSE
p2$cores <- 1

dL <- RandomForestDeployment$new(p2)
dL$deploy()

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

## 1. Loading data and packages.
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]
FROM [SAM].[dbo].[HCRDiabetesClinical]
"

df <- selectData(connection.string, query)

# Save a dataframe for validation later on
dfDeploy <- df[951:1000,]

## 2. Train and save the model using DEVELOP
print('Historical, development data:')
str(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 RandomForest
RandomForest <- RandomForestDevelopment$new(p)
RandomForest$run()

## 3. Load saved model and use DEPLOY to generate predictions.
print('Fake production data:')
str(dfDeploy)

p2 <- SupervisedModelDeploymentParams$new()
p2$type <- "classification"
p2$df <- dfDeploy
p2$grainCol <- "PatientEncounterID"
p2$predictedCol <- "ThirtyDayReadmitFLG"
p2$impute <- TRUE
p2$debug <- FALSE
```

```
p2$cores <- 1

dL <- RandomForestDeployment$new(p2)
dL$deploy()
dfOut <- dL$getOutDf()

writeData(MSSQLConnectionString = connection.string,
          df = dfOut,
          tableName = 'HCRDeployClassificationBASE')

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)
# )

## 1. Loading data and packages.
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]
FROM [SAM].[dbo].[HCRDiabetesClinical]
"

df <- selectData(connection.string, query)

# Save a dataframe for validation later on
dfDeploy <- df[951:1000,]

## 2. Train and save the model using DEVELOP
print('Historical, development data:')
```

```
str(df)

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 Random Forest
RandomForest <- RandomForestDevelopment$new(p)
RandomForest$run()

## 3. Load saved model and use DEPLOY to generate predictions.
dfDeploy$A1CNBR <- NULL # You won't know the response in production
print('Fake production data:')
str(dfDeploy)

p2 <- SupervisedModelDeploymentParams$new()
p2$type <- "regression"
p2$df <- dfDeploy
p2$grainCol <- "PatientEncounterID"
p2$predictedCol <- "A1CNBR"
p2$impute <- TRUE
p2$debug <- FALSE
p2$cores <- 1

dL <- RandomForestDeployment$new(p2)
dL$deploy()
dfOut <- dL$getOutDf()

writeData(MSSQLConnectionString = connection.string,
          df = dfOut,
          tableName = 'HCRDeployRegressionBASE')

print(proc.time() - ptm)


#' #### Classification example pulling from CSV and writing to SQLite ####

## 1. Loading data and packages.
ptm <- proc.time()
library(healthcareai)

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

sqliteFile <- system.file("extdata",
```

```
                          "unit-test.sqlite",
                          package = "healthcareai")

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

df$PatientID <- NULL # Only one ID column (ie, PatientEncounterID) is needed remove this column

# Save a dataframe for validation later on
dfDeploy <- df[951:1000,]

## 2. Train and save the model using DEVELOP
print('Historical, development data:')
str(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 Random Forest
RandomForest <- RandomForestDevelopment$new(p)
RandomForest$run()

## 3. Load saved model and use DEPLOY to generate predictions.
print('Fake production data:')
str(dfDeploy)

p2 <- SupervisedModelDeploymentParams$new()
p2$type <- "classification"
p2$df <- dfDeploy
p2$grainCol <- "PatientEncounterID"
p2$predictedCol <- "ThirtyDayReadmitFLG"
p2$impute <- TRUE
p2$debug <- FALSE
p2$cores <- 1

dL <- RandomForestDeployment$new(p2)
dL$deploy()
dfOut <- dL$getOutDf()

writeData(SQLiteFileName = sqliteFile,
          df = dfOut,
          tableName = 'HCRDeployClassificationBASE')

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

```
#### Regression example pulling from CSV and writing to SQLite ####

## 1. Loading data and packages.
ptm <- proc.time()
library(healthcareai)

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

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

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

df$PatientID <- NULL # Only one ID column (ie, PatientEncounterID) is needed remove this column

# Save a dataframe for validation later on
dfDeploy <- df[951:1000,]

## 2. Train and save the model using DEVELOP
print('Historical, development data:')
str(df)

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 Random Forest
RandomForest<- RandomForestDevelopment$new(p)
RandomForest$run()

## 3. Load saved model and use DEPLOY to generate predictions.
dfDeploy$A1CNBR <- NULL # You won't know the response in production
print('Fake production data:')
str(dfDeploy)

p2 <- SupervisedModelDeploymentParams$new()
p2$type <- "regression"
p2$df <- dfDeploy
p2$grainCol <- "PatientEncounterID"
```

```
p2$predictedCol <- "A1CNBR"
p2$impute <- TRUE
p2$debug <- FALSE
p2$cores <- 1

dL <- RandomForestDeployment$new(p2)
dL$deploy()
dfOut <- dL$getOutDf()

writeData(SQLiteFileName = sqliteFile,
          df = dfOut,
          tableName = 'HCRDeployRegressionBASE')

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. Random forest is an ensemble model, well suited for non-linear data. It's fast to train and often a good starting point.

Usage

```
RandomForestDevelopment(type, df, grainCol, predictedCol,
impute, debug, cores, trees, tune, modelName)
```

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.
`predictedCol`	Column that you want to predict. If you're doing classification then this should be Y/N.
`impute`	Set all-column imputation to T or F. If T, this uses mean replacement for numeric columns and most frequent for factorized columns. F leads to removal of rows containing NULLs. Values are saved for deployment.
`debug`	Provides the user extended output to the console, in order to monitor the calculations throughout. Use T or F.
`cores`	Number of cores you'd like to use. Defaults to 2.
`trees`	Number of trees in the forest. Defaults to 201.
`tune`	If TRUE, automatically tune model for better performance. Creates grid using mtry param and 5-fold cross validation. Note this takes longer.
`modelName`	Optional string. Can specify the model name. If used, you must load the same one in the deploy step.

Format

An object of class `R6ClassGenerator` of length 24.

Methods

The above describes params for initializing a new randomForestDevelopment class with `$new()`. Individual methods are documented below.

`$new()`

Initializes a new random forest development class using the parameters saved in `p`, documented above. This method loads, cleans, and prepares data for model training.
Usage: `$new(p)`

`$run()`

Trains model, displays feature importance and performance.
Usage:`$new()`

`$getPredictions()`

Returns the predictions from test data.
Usage: `$getPredictions()`

`$getROC()`

Returns the ROC curve object for `plotROCs`. Classification models only.
Usage: `$getROC()`

`$getPRCurve()`

Returns the PR curve object for `plotPRCurve`. Classification models only.
Usage: `$getROC()`

`$getAUROC()`

Returns the area under the ROC curve from testing for classification models.
Usage: `$getAUROC()`

`$getRMSE()`

Returns the RMSE from test data for regression models.
Usage: `$getRMSE()`

`$getMAE()`

Returns the RMSE from test data for regression models.
Usage: `$getMAE()`

`$getVariableImportanceList()`

Returns the variable importance list.
Usage: `$getVariableImportanceList(numTopVariables = NULL)`
Params:
- `numTopVariables`: The maximum number of variables to include. If no value is specified, all variables are used.

`$plotVariableImportance()`

Plots the variable importance list.
Usage: `$plotVariableImportance(numTopVariables = NULL)`
Params:
- `numTopVariables`: The maximum number of variables to include. If no value is specified, all variables are used.

References

http://healthcareai-r.readthedocs.io

See Also

LassoDevelopment

LinearMixedModelDevelopment

selectData

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

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
"

# This query should pull only rows for training. They must have a label.
query <- "
SELECT
[PatientEncounterID]
,[SystolicBPNBR]
,[LDLNBR]
,[A1CNBR]
,[GenderFLG]
,[ThirtyDayReadmitFLG]
FROM [SAM].[dbo].[HCRDiabetesClinical]
"

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)
```

`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://healthcareai-r.readthedocs.io

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)
```

`removeColsWithDTSSuffix` *Remove columns with DTS suffix*

Description

Remove columns with DTS in the suffix of the column name

Usage

```
removeColsWithDTSSuffix(df)
```

Arguments

`df`	A data frame to be altered

Value

The input data frame with DTS suffix columns removed

References

http://healthcareai-r.readthedocs.io

See Also

healthcareai

Examples

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

`removeColsWithOnlyNA` *Remove columns from a data frame that are only NA*

Description

Remove columns from a data frame when all of their rows are NA's

Usage

```
removeColsWithOnlyNA(df)
```

Arguments

`df`	A data frame

Value

A data frame with columns of only NA's 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','NA','a'),
                 d=c(NA,NA,NA))
dfResult <- removeColsWithOnlyNA(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://healthcareai-r.readthedocs.io

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://healthcareai-r.readthedocs.io

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://healthcareai-r.readthedocs.io

See Also

healthcareai

Examples

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

## Not run:
library(healthcareai)

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

query <- "
SELECT
[PatientEncounterID]
,[PatientID]
,[SystolicBPNBR]
,[LDLNBR]
,[A1CNBR]
,[GenderFLG]
,[ThirtyDayReadmitFLG]
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()

## End(Not 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)
```

Arguments

MSSQLConnectionString	A string specifying the driver, server, database, and whether Windows Authentication will be used.
query	The SQL query (in ticks or quotes)
SQLiteFileName	A string. If dbtype is SQLite, here one specifies the database file to query from
randomize	Boolean that dictates whether returned rows are randomized

Value

df A data frame containing the selected rows

References

http://healthcareai-r.readthedocs.io

See Also

healthcareai

writeData

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)
```

`skip_if_no_MSSQL` *Function to skip specific tests if MSSQL/specific databases not set up on user's machine.*

Description

This function is used in the testing files where R is used to write to MSSQL. It first checks for a connection to MSSQL. If no connection to MSSQL, it will skip the test. If there is a good connection but the specific database is not present it will skip the test as well.

Usage

```
skip_if_no_MSSQL(tableName, connString)
```

Arguments

`tableName`	What table to look for if connection is good. Entered as a string.
`connString`	Connection string to use when trying to connect to MSSQL

References

http://healthcareai-r.readthedocs.io

See Also

healthcareai

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://healthcareai-r.readthedocs.io

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://healthcareai-r.readthedocs.io

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://healthcareai-r.readthedocs.io

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://healthcareai-r.readthedocs.io

See Also

healthcareai

`UnsupervisedModel` *Build clusters based on your data.*

Description

This step allows one to build clusters on your data

Usage

```
UnsupervisedModel(object)
```

Arguments

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

Format

An object of class `R6ClassGenerator` of length 24.

References

http://healthcare.ai

See Also

healthcareai

KmeansClustering

`UnsupervisedModelParams` *UnsupervisedModelParams class to set up parameters required to build UnsupervisedModel classes*

Description

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

Usage

```
UnsupervisedModelParams
```

Format

An object of class `R6ClassGenerator` of length 24.

References

http://healthcare.ai

See Also

healthcareai

KmeansClustering

`variationAcrossGroups` *Find variation across groups*

Description

Compare variation among groups on a continuous measure. Plot boxplot and perform ANOVA with Tukey's HSD statistical test to compare variation in a numeric variable (measureColumn) across groups (categoricalCols).

Usage

```
variationAcrossGroups(df, categoricalCols, measureColumn,
  plotGroupDifferences = FALSE, returnGroupStats = FALSE, dateCol = NULL,
  levelOfDateGroup = "monthly", sigLevel = 0.05)
```

Arguments

df	A data frame containing group and measure columns.
categoricalCols	Character. Vector containing the name(s) of column(s) to group by.
measureColumn	Character. The name of the numeric variable of interest.
plotGroupDifferences	Optional. Logical. Plot results of Tukey's HSD test: mean differences between groups and confidence intervals for each pairwise group comparison? Default is FALSE.
returnGroupStats	Optional. Logical. In addition to the model summary table, return summary statistics for each group? Default is FALSE
dateCol	Optional. Date. A date(time) column to group by (grouped by month by default).
levelOfDateGroup	Optional. Character. Level at which to group dateCol. One of "yearly", "quarterly", "monthly" (default), or "weekly".
sigLevel	Optional. Numeric value between zero and one giving the alpha value for Tukey HSD test, i.e. the p-value threshold for significance.

Value

By default, a data frame giving summary statistics from Tukey's HSD test. If returnGroupStats is TRUE, a list of two data frames giving model and group summary statistics respectively.

This function always induces the side effect of printing a boxplot to compare variation across groups. If plotGroupDifferences is TRUE, also plots a mean differences and confidence intervals between groups.

References

http://healthcare.ai

This function uses multcompView::multcompLetters() https://CRAN.R-project.org/package=multcompView

See Also

healthcareai findVariation

Examples

```
### Example 1 ###

# This example dataset has two columns: a blood test result (value)
# and an anesthetic treatment (anesthetic). There are five anesthetics in the
# dataset: Midazolam, Propofol, Ketamine, Thiamylal, and Diazepam.

set.seed(35)
```

```
df1 <- data.frame(
  anesthetic = c(rep("Midazolam", 50), rep("Propofol", 20), rep("Ketamine", 40),
                 rep("Thiamylal", 80),  rep("Diazepam", 20)),
  value = c(sample(2:5, 50, replace = TRUE), sample(6:10, 20, replace = TRUE),
            sample(1:7, 40, replace = TRUE), sample(3:10, 80, replace = TRUE),
            sample(10:20, 20, replace = TRUE)))
head(df1)

variationAcrossGroups(df1, "anesthetic", "value", sigLevel = .01)

# The boxplot tells us that Diazepam, Propofol, and Thiamylal all have
# significantly different mean values from all other groups, including each other
# (p <= 0.01). Midazolam and Ketamine do not have significantly different mean
# values because they share the label "a", but they are significantly different
# from all the other treatments.



### Example 2 ###

# This example dataset has three columns: department, admission date, and
# blood pressure reading (BP). We will examine whether blood pressures
# vary by department and year of admission.

set.seed(2017)
n <- 200
bp <- data.frame(department = sample(c("Cardiology", "Oncology", "Gastroenterology"),
                                     size = n,
                                     replace = TRUE,
                                     prob =  c(0.5, 0.3, 0.2)),
                 admit_date = sample(seq(as.Date("2015-01-01"),
                                         as.Date("2017-12-31"),
                                         by = "day"),
                                     size = n))
bp$BP <- floor(rnorm(n,
                     120 +
                       ifelse(bp$admit_date > "2015-12-31", 1.5, 1) +
                       ifelse(bp$department == "Cardiology", 80, 0),
                     ifelse(bp$department == "Oncology", 60, 30)))
head(bp)

variationAcrossGroups(bp,
                      categoricalCols = "department",
                      measureColumn = "BP",
                      dateCol = "admit_date",
                      levelOfDateGroup = "yearly",
                      plotGroupDifferences = TRUE)

# Since plotGroupDifferences = TRUE and the default of returnGroupStats is
# FALSE, the function prints the boxplot and the 95% family-wise confidence
# interval plot, and returns the summary statistics data frame. The two plots
# show:
#
```

```
#   1. The boxplot of BP across all combinations of the two categories.
#   department has 3 levels, as does date grouped by year, so there are a total
#   of 3 x 3 = 9 groupings, which are shown on the x axis of the boxplot. Groups
#   that have a shared letter are *not* significantly different. For example,
#   (Gastroenterology | 2015) and (Oncology | 2015) share a "b" label, so
#   patients in those groups do not have significantly different mean BP. On the
#   other hand, (Cardiology | 2015) and (Gastroenterology | 2015) do not share a
#   label, so patients in those groups do have significantly different BP. Likewise,
#  Oncology patients in 2015 have different BPs from Oncology patients in either of the subsequent
#   years, but Oncology patients in 2016 and 2017 do not have significantly
#   different BP (as shown by their shared "c" label).
#
#   2. Confidence-interval level plot This plot present the results
#   of the Tukey's Honest Significant Differences test. It compares all possible
#   pairs of groups and adjusts p-values for multiple comparisons. Red lines
#   indicate a significant difference between the two groups at the chosen
#   significance level (0.05 by default). Groups are ordered by p-values. The
#   group with the greater mean value is always listed first (e.g. Cardiology |
#   2016 has greater BP than Oncology 2015).
```

`writeData` *Write data to database*

Description

Write data frame to database table via ODBC connection

Usage

```
writeData(MSSQLConnectionString = NULL, df, SQLiteFileName = NULL,
  tableName, addSAMUtilityColumns = FALSE, connectionString = NULL)
```

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
`addSAMUtilityColumns`	Boolean. Whether to add Health Catalyst-related date-time stamp, BindingID, and BindingNM to df before saving to db.
`connectionString`	Deprecated. A string specifying the driver, server, database, and whether Windows Authentication will be used. See ?MSSQLConnectionString instead.

Value

Nothing

References

http://healthcareai-r.readthedocs.io

See Also

healthcareai

selectData

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')


## Not run:
#This example shows the RODBC way of writing to a non-default schema while
#ODBC is being fixed. Here is a link to the non-default issue in ODBC:
#https://github.com/rstats-db/odbc/issues/91
```

```
#First, create this table in SQL Server using a non-default schema. The
#example creates this table in the SAM database on localhost. You will also
#need to create a new schema(Cardiovascular) in SSMS for this specific
#example to work.
#CREATE TABLE [Cardiovascular].[TestTable](
#[a] [float] NULL,
#[b] [float] NULL,
#[c] [varchar](255) NULL)

# Install the RODBC pacakge onto your machine. You only need to do this one
# time.
#install.packages("RODBC")

# Load the package
library(RODBC)

# Create a connection to work with
con <- RODBC::odbcDriverConnect('driver={SQL Server};
                                 server=localhost;
                                 database=SAM;
                                 trusted_connection=true')

# Df write to SQL Server. df columns names must match the SQL table in SSMS.
df <- data.frame(a = c(10, 20, 30),
                 b = c(20, 40, 60),
                 c = c("oneT", "twoT", "threeT"))

# Write the df to the SQL table
RODBC::sqlSave(con, df, "Cardiovascular.TestTable", append = TRUE,
               rownames = FALSE)

# Verify that the table was written to
confirmDf <- RODBC::sqlQuery(con, 'select * from Cardiovascular.TestTable')
head(confirmDf)

## End(Not run)
```

`XGBoostDeployment` *Deploy a production-ready predictive XGBoost model*

Description

This step allows one to

- Automatically load a saved model from `XGBoostDevelopment`
- Run the model against test data to generate predictions
- Push these predictions to SQL Server or CSV

Usage

```
XGBoostDeployment(type, df, grainCol,
predictedCol, impute, debug, cores, modelName)
```

Arguments

type	The type of model (must be multiclass)
df	Dataframe whose columns are used for new predictions. Data structure should match development as much as possible. Number of columns, names, types, grain, and predicted must be the same.
grainCol	The dataframe's column that has IDs pertaining to the grain
predictedCol	Column that you want to predict.
impute	For training df, set all-column imputation to T or F. If T, this uses values calculated in development. F leads to removal of rows containing NULLs and is not recommended.
debug	Provides the user extended output to the console, in order to monitor the calculations throughout. Use T or F.
cores	Number of cores you'd like to use. Defaults to 2.
modelName	Optional string. Can specify the model name. If used, you must load the same one in the deploy step.

Format

An object of class `R6ClassGenerator` of length 24.

Methods

The above describes params for initializing a new XGBoostDeployment class with `$new()`. Individual methods are documented below.

`$new()`

Initializes a new XGBoost deployment class using the parameters saved in `p`, documented above. This method loads, cleans, and prepares data for generating predictions.
Usage: `$new(p)`

`$deploy()`

Generate new predictions and prepare the output dataframe.
Usage: `$deploy()`

`$getPredictions()`

Return the grain and predictions for each class.
Usage: `$getPredictions()`

`$getOutDf()`

Returns a dataframe containing the grain column, the top 3 probabilities for each row, and the classes associated with those probabilities.
Usage: `$getOutDf()`

See Also

healthcareai

writeData

selectData

Examples

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

# 1. Load data. Categorical columns should be characters.
# can delete these system.file lines in your work
csvfile <- system.file("extdata",
                       "dermatology_multiclass_data.csv",
                       package = "healthcareai")
# Read in CSV; replace csvfile with 'path/file'
df <- read.csv(file = csvfile,
               header = TRUE,
              stringsAsFactors = FALSE,
               na.strings = c("NULL", "NA", "", "?"))

str(df) # check the types of columns
dfDeploy <- df[347:366,] # reserve 20 rows for deploy step.

# 2. Develop and save model (saving is automatic)
set.seed(42)
p <- SupervisedModelDevelopmentParams$new()
p$df <- df
p$type <- "multiclass"
p$impute <- TRUE
p$grainCol <- "PatientID"
p$predictedCol <- "target"
p$debug <- FALSE
p$cores <- 1
# xgb_params must be a list with all of these things in it.
# if you would like to tweak parameters, go for it!
# Leave objective and eval_metric as they are.
p$xgb_params <- list("objective" = "multi:softprob",
                     "eval_metric" = "mlogloss",
                     "max_depth" = 6, # max depth of each learner
                     "eta" = 0.1, # learning rate
                     "silent" = 0, # verbose output when set to 1
                     "nthread" = 2) # number of processors to use
```

```
# Run model
boost <- XGBoostDevelopment$new(p)
boost$run()

## 3. Load saved model (automatic) and use DEPLOY to generate predictions.
p2 <- SupervisedModelDeploymentParams$new()
p2$type <- "multiclass"
p2$df <- dfDeploy
p2$grainCol <- "PatientID"
p2$predictedCol <- "target"
p2$impute <- TRUE
p2$debug <- FALSE

# Deploy model to make new predictions
boostD <- XGBoostDeployment$new(p2)
boostD$deploy()

# Get output dataframe for csv or SQL
outDf <- boostD$getOutDf()
head(outDf)

# If you want to write to sqlite:
# sqliteFile <- system.file("extdata",
#                           "unit-test.sqlite",
#                          package = "healthcareai")
# writeData(SQLiteFileName = sqliteFile,
#         df = outDf,
#         tableName = "dermatologyDeployMulticlassBASE")

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

# Get raw predictions if you want
# rawPredictions <- boostD$getPredictions()

# If you have known labels, check your prediction accuracy like this:
# caret::confusionMatrix(true_label,
#              predicted_label,
#              mode = "everything")

print(proc.time() - ptm)


#### Example pulling from CSV and writing to SQL server ####
# 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].[dermatologyDeployClassificationBASE](
# [BindingID] [int] NULL,[BindingNM] [varchar](255) NULL,
# [LastLoadDTS] [datetime2](7) NULL,
# [PatientID] [decimal](38, 0) NULL,
# [PredictedProb1] [decimal](38, 2) NULL,
# [PredictedClass1] [varchar](255) NULL,
# [PredictedProb2] [decimal](38, 2) NULL,
```

```
# [PredictedClass2] [varchar](255) NULL,
# [PredictedProb3] [decimal](38, 2) NULL,
# [PredictedClass3] [varchar](255) NULL)


# 1. Load data. Categorical columns should be characters.
csvfile <- system.file("extdata",
                       "dermatology_multiclass_data.csv",
                       package = "healthcareai")

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

str(df) # check the types of columns
dfDeploy <- df[347:366,] # reserve 20 rows for deploy step.


# 2. Develop and save model (saving is automatic)
set.seed(42)
p <- SupervisedModelDevelopmentParams$new()
p$df <- df
p$type <- "multiclass"
p$impute <- TRUE
p$grainCol <- "PatientID"
p$predictedCol <- "target"
p$debug <- FALSE
p$cores <- 1
# xgb_params must be a list with all of these things in it.
# if you would like to tweak parameters, go for it!
# Leave objective and eval_metric as they are.
p$xgb_params <- list("objective" = "multi:softprob",
                     "eval_metric" = "mlogloss",
                     "max_depth" = 6, # max depth of each learner
                     "eta" = 0.1, # learning rate
                     "silent" = 0, # verbose output when set to 1
                     "nthread" = 2) # number of processors to use

# Run model
boost <- XGBoostDevelopment$new(p)
boost$run()

## 3. Load saved model (automatic) and use DEPLOY to generate predictions.
p2 <- SupervisedModelDeploymentParams$new()
p2$type <- "multiclass"
p2$df <- dfDeploy
p2$grainCol <- "PatientID"
p2$predictedCol <- "target"
p2$impute <- TRUE
p2$debug <- FALSE
```

```
# Deploy model to make new predictions
boostD <- XGBoostDeployment$new(p2)
boostD$deploy()

# Get output dataframe for csv or SQL
outDf <- boostD$getOutDf()
head(outDf)

# Save the output to SQL server

connection.string <- "
driver={SQL Server};
server=localhost;
database=SAM;
trusted_connection=true
"
writeData(MSSQLConnectionString = connection.string,
       df = outDf,
       tableName = 'dermatologyDeployClassificationBASE')

# Get raw predictions if you want
# rawPredictions <- boostD$getPredictions()
print(proc.time() - ptm)
```

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

Description

This step allows you to create an XGBoost classification model, based on your data. Use model type 'multiclass' with 2 or more classes. XGBoost is an ensemble model, well suited to non-linear data and very fast. Can be parameter-dependent.

Usage

```
XGBoostDevelopment(type, df, grainCol, predictedCol,
impute, debug, cores, modelName, xgb_params, xgb_nrounds)
```

Arguments

`type`	The type of model. Currently requires 'multiclass'.
`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 T or F. If T, this uses mean replacement for numeric columns and most frequent for factorized columns. F leads to removal of rows containing NULLs. Values are saved for deployment.
debug	Provides the user extended output to the console, in order to monitor the calculations throughout. Use T or F.
cores	Number of cores you'd like to use. Defaults to 2.
modelName	Optional string. Can specify the model name. If used, you must load the same one in the deploy step.
xgb_params	A list, containing optional xgboost parameters. The full list of params can be found at http://xgboost.readthedocs.io/en/latest/parameter.html.
xgb_nrounds	Number of rounds to use for boosting.

Format

An object of class R6ClassGenerator of length 24.

Methods

The above describes params for initializing a new XGBoostDevelopment class with $new(). Individual methods are documented below.

$new()

Initializes a new XGBoost development class using the parameters saved in p, documented above. This method loads, cleans, and prepares data for model training.
Usage: $new(p)

$run()

Trains model, displays predictions and class-wise performance.
Usage: $new()

$getPredictions()

Returns the predictions from test data.
Usage: $getPredictions()

$generateConfusionMatrix()

Returns the confusion matrix and statistics generated during model development.
Usage: $getConfusionMatrix()

References

http://healthcareai-r.readthedocs.io

See Also

Information on the example dataset can be found at: http://archive.ics.uci.edu/ml/datasets/dermatology/

Information on the xgboost parameters can be found at: https://github.com/dmlc/xgboost/blob/master/doc/parameter.md

selectData

Examples

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

# 1. Load data. Categorical columns should be characters.
csvfile <- system.file("extdata",
                       "dermatology_multiclass_data.csv",
                       package = "healthcareai")

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

str(df) # check the types of columns

# 2. Develop and save model
set.seed(42)
p <- SupervisedModelDevelopmentParams$new()
p$df <- df
p$type <- "multiclass"
p$impute <- TRUE
p$grainCol <- "PatientID"
p$predictedCol <- "target"
p$debug <- FALSE
p$cores <- 1
# xgb_params must be a list with all of these things in it.
# if you would like to tweak parameters, go for it!
# Leave objective and eval_metric as they are.
p$xgb_params <- list("objective" = "multi:softprob",
                     "eval_metric" = "mlogloss",
                     "max_depth" = 6, # max depth of each learner
                     "eta" = 0.1, # learning rate
                     "silent" = 0, # verbose output when set to 1
                     "nthread" = 2) # number of processors to use

# Run model
boost <- XGBoostDevelopment$new(p)
boost$run()
```

```
# Get output data
outputDF <- boost$getPredictions()
head(outputDF)

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

Index