Package ‘RemixAutoML’

June 14, 2019

Title Remix Automated Machine Learning

Version 1.5.0

Date 2019-06-03

Maintainer Adrian Antico <adrianantico@gmail.com>

Description Automates and ensures high quality output for most of your machine learning and data science tasks. The package contains high quality functions that run at efficient speed with minimal memory constraints for supervised learning, unsupervised learning, feature engineering, model evaluation and interpretation, along with some helper functions for graphing. AutoCatBoostClassifier(), AutoCatBoostRegression(), and AutoCatBoostMultiClass() have a dependency to the catboost package which isn't part of the CRAN repository at the time of this writing. The link to the catboost URL to download the package for use is in the Additional_repositories field below, which has the installation instructions. You need to install that package to make use of the AutoCatBoost_ functions.

License MPL-2.0

URL https://github.com/AdrianAntico/RemixAutoML

BugReports https://github.com/AdrianAntico/RemixAutoML/issues

Depends R (¿= 3.5.0)

Imports catboost,
caTools,
data.table,
doParallel,
foreach,
forecast,
ggplot2,
grid,
h2o,
itertools,
lubridate,
magick,
methods,
monreg,
parallel,
pROC,
RColorBrewer,

recommenderlab,
ROCR,
scatterplot3d,
stats,
stringr,
tm,
tsoutliers,
utils,
wordcloud,
xgboost,
zoo

Suggests knitr,
rmarkdown,
sde,
testthat

VignetteBuilder knitr

Additional_repositories
https://github.com/catboost/catboost/tree/master/catboost/R-package

Contact Adrian Antico

Encoding UTF-8

Language en-US

LazyData true

NeedsCompilation no

RoxygenNote 6.1.1

SystemRequirements Java (¿= 7.0)

R topics documented:

`AutoCatBoostClassifier` *AutoCatBoostClassifier is an automated catboost model grid-tuning classifier and evaluation system*

Description

AutoCatBoostClassifier is an automated modeling function that runs a variety of steps. First, a stratified sampling (by the target variable) is done to create train, validation, and test sets (if not supplied). Then, the function will run a random grid tune over N number of models and find which model is the best (a default model is always included in that set). Once the model is identified and built, several other outputs are generated: validation data with predictions (on test data), an ROC plot, evaluation plot, evaluation metrics, variable importance, partial dependence calibration plots, partial dependence calibration box plots, and column names used in model fitting. You can download the catboost package using devtools, via: devtools::install_github('catboost/catboost', subdir = 'catboost/R-package')

Usage

```
AutoCatBoostClassifier(data, ValidationData = NULL, TestData = NULL,
  TargetColumnName = NULL, FeatureColNames = NULL,
  PrimaryDateColumn = NULL, ClassWeights = NULL, IDcols = NULL,
  task_type = "GPU", eval_metric = "AUC", Trees = 50,
  GridTune = FALSE, grid_eval_metric = "f", MaxModelsInGrid = 10,
  model_path = NULL, ModelID = "FirstModel", NumOfParDepPlots = 3,
  ReturnModelObjects = TRUE, SaveModelObjects = FALSE,
  PassInGrid = NULL)
```

Arguments

`data` This is your data set for training and testing your model

`ValidationData` This is your holdout data set used in modeling either refine your hyperparameters. Catboost using both training and validation data in the training process so you should evaluate out of sample performance with this data set.

`TestData` This is your holdout data set. Catboost using both training and validation data in the training process so you should evaluate out of sample performance with this data set.

`TargetColumnName` Either supply the target column name OR the column number where the target is located, but not mixed types. Note that the target column needs to be a 0 — 1 numeric variable.

`FeatureColNames` Either supply the feature column names OR the column number where the target is located, but not mixed types. Also, not zero-indexed.

`PrimaryDateColumn` Supply a date or datetime column for catboost to utilize time as its basis for handling categorical features, instead of random shuffling

`ClassWeights` Supply a vector of weights for your target classes. E.g. c(0.25, 1) to weight your 0 class by 0.25 and your 1 class by 1.

`IDcols` A vector of column names or column numbers to keep in your data but not include in the modeling.

`task_type` "GPU" Set to "GPU" to utilize your GPU for training. Default is "CPU".

`eval_metric` This is the metric used inside catboost to measure performance on validation data during a grid-tune. "AUC" is the default, but other options include "Logloss", "CrossEntropy", "Precision", "Recall", "F1", "BalancedAccuracy", "BalancedErrorRate", "MCC", "Accuracy", "CtrFactor", "AUC", "BrierScore", "HingeLoss", "HammingLoss", "ZeroOneLoss", "Kappa", "WKappa", "LogLikelihoodOfPrediction"

`Trees` The maximum number of trees you want in your models

`GridTune` Set to TRUE to run a grid tuning procedure. Set a number in MaxModelsInGrid to tell the procedure how many models you want to test.

`grid_eval_metric` This is the metric used to find the threshold "f", "auc", "tpr", "fnr", "fpr", "tnr", "prbe", "f", "odds"

`MaxModelsInGrid` Number of models to test from grid options. 1080 total possible options

model_path
: A character string of your path file to where you want your output saved

ModelID
: A character string to name your model and output

NumOfParDepPlots
: Tell the function the number of partial dependence calibration plots you want to create. Calibration boxplots will only be created for numerical features (not dummy variables)

ReturnModelObjects
: Set to TRUE to output all modeling objects. E.g. plots and evaluation metrics

SaveModelObjects
: Set to TRUE to return all modeling objects to your environment

PassInGrid
: Defaults to NULL. Pass in a single row of grid from a previous output as a data.table (they are collected as data.tables)

Value

Saves to file and returned in list: VariableImportance.csv, Model (the model), ValidationData.csv, ROC_Plot.png, EvalutionPlot.png, EvaluationMetrics.csv, ParDepPlots.R a named list of features with partial dependence calibration plots, GridCollect, and GridList

Author(s)

Adrian Antico

See Also

Other Supervised Learning: AutoCatBoostMultiClass, AutoCatBoostRegression, AutoCatBoostScoring, AutoH2OMLScoring, AutoH2OModeler, AutoH2OScoring, AutoH2oDRFClassifier, AutoH2oDRFMultiClass, AutoH2oDRFRegression, AutoH2oGBMClassifier, AutoH2oGBMMultiClass, AutoH2oGBMRegression, AutoMLTS, AutoNLS, AutoRecommenderScoring, AutoRecommender, AutoTS, AutoXGBoostClassifier, AutoXGBoostMultiClass, AutoXGBoostRegression, AutoXGBoostScoring

Examples

```
Correl <- 0.85
N <- 1000
data <- data.table::data.table(Target = runif(N))
data[, x1 := qnorm(Target)]
data[, x2 := runif(N)]
data[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable2 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable3 := exp(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                                sqrt(1-Correl^2) * qnorm(x2))))]
data[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                             sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable6 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.10]
data[, Independent_Variable7 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.25]
```

```
data[, Independent_Variable8 := (pnorm(Correl * x1 +
                                        sqrt(1-Correl^2) * qnorm(x2)))^0.75]
data[, Independent_Variable9 := (pnorm(Correl * x1 +
                                        sqrt(1-Correl^2) * qnorm(x2)))^2]
data[, Independent_Variable10 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^4]
data[, Independent_Variable11 := as.factor(
  ifelse(Independent_Variable2 < 0.20, "A",
         ifelse(Independent_Variable2 < 0.40, "B",
                ifelse(Independent_Variable2 < 0.6,  "C",
                       ifelse(Independent_Variable2 < 0.8,  "D", "E")))))]
data[, ':=' (x1 = NULL, x2 = NULL)]
data[, Target := ifelse(Target < 0.5, 1, 0)]
TestModel <- AutoCatBoostClassifier(data,
                                    ValidationData = NULL,
                                    TestData = NULL,
                                    TargetColumnName = "Target",
                                    FeatureColNames = c(2:12),
                                    PrimaryDateColumn = NULL,
                                    ClassWeights = NULL,
                                    IDcols = NULL,
                                    MaxModelsInGrid = 3,
                                    task_type = "GPU",
                                    eval_metric = "AUC",
                                    grid_eval_metric = "auc",
                                    Trees = 50,
                                    GridTune = FALSE,
                                    model_path = NULL,
                                    ModelID = "ModelTest",
                                    NumOfParDepPlots = 15,
                                    ReturnModelObjects = TRUE,
                                    SaveModelObjects = FALSE,
                                    PassInGrid = NULL)
```

`AutoCatBoostdHurdleModel` *AutoCatBoostdHurdleModel is a Retrain Function for the Regression Models for the Subsetted Data in P6*

Description

AutoCatBoostdHurdleModel is a Retrain Function for the Regression Models for the Subsetted Data in P6

AutoCatBoostdHurdleModel is a Retrain Function for the Regression Models for the Subsetted Data in P6

Usage

```
AutoCatBoostdHurdleModel(data, ValidationData = NULL, TestData = NULL,
  Buckets = c(1, 5, 10, 20), TargetColumnName = "Target",
  FeatureColNames = 4:ncol(data), PrimaryDateColumn = NULL,
  IDcols = NULL, ClassWeights = NULL, SplitRatios = c(0.7, 0.2, 0.1),
```

```
  task_type = "GPU", ModelID = "ModelTest", Paths = NULL,
  SaveModelObjects = TRUE, Trees = 15000, GridTune = TRUE,
  MaxModelsInGrid = 1, NumOfParDepPlots = 10, PassInGrid = NULL)

AutoCatBoostdHurdleModel(data, ValidationData = NULL, TestData = NULL,
  Buckets = c(1, 5, 10, 20), TargetColumnName = "Target",
  FeatureColNames = 4:ncol(data), PrimaryDateColumn = NULL,
  IDcols = NULL, ClassWeights = NULL, SplitRatios = c(0.7, 0.2, 0.1),
  task_type = "GPU", ModelID = "ModelTest", Paths = NULL,
  SaveModelObjects = TRUE, Trees = 15000, GridTune = TRUE,
  MaxModelsInGrid = 1, NumOfParDepPlots = 10, PassInGrid = NULL)
```

Arguments

data	Source training data. Do not include a column that has the class labels for the buckets as they are created internally.
ValidationData	Source validation data. Do not include a column that has the class labels for the buckets as they are created internally.
TestData	Souce test data. Do not include a column that has the class labels for the buckets as they are created internally.
Buckets	A numeric vector of the buckets used for subsetting the data. NOTE: the final Bucket value will first create a subset of data that is less than the value and a second one thereafter for data greater than the bucket value.
TargetColumnName	Supply the column name or number for the target variable
FeatureColNames	Supply the column names or number of the features (not included the PrimaryDateColumn)
PrimaryDateColumn	Supply a date column if the data is functionally related to it
IDcols	Includes PrimaryDateColumn and any other columns you want returned in the validation data with predictions
ClassWeights	Utilize these for the classifier model
SplitRatios	Supply vector of partition ratios. For example, c(0.70,0.20,0,10).
task_type	Set to "GPU" or "CPU"
ModelID	Define a character name for your models
Paths	A character vector of the path file strings. EITHER SUPPLY 1 file path or N file paths for N models
SaveModelObjects	Set to TRUE to save the model objects to file in the folders listed in Paths
Trees	Default 15000
GridTune	Set to TRUE if you want to grid tune the models
NumOfParDepPlots	Set to pull back N number of partial dependence calibration plots.
PassInGrid	Pass in a grid for changing up the parameter settings for catboost
RegressionModels	Set to the model of choice. Currently only catboost is available.

`ClassificationModels`	Set to the model of choice. Currently, only catboost is available.
`NumberModelsInGrid`	Set to a numeric value for the number of models to try in grid tune
`data`	Source training data. Do not include a column that has the class labels for the buckets as they are created internally.
`ValidationData`	Source validation data. Do not include a column that has the class labels for the buckets as they are created internally.
`TestData`	Souce test data. Do not include a column that has the class labels for the buckets as they are created internally.
`Buckets`	A numeric vector of the buckets used for subsetting the data. NOTE: the final Bucket value will first create a subset of data that is less than the value and a second one thereafter for data greater than the bucket value.
`TargetColumnName`	Supply the column name or number for the target variable
`FeatureColNames`	Supply the column names or number of the features (not included the PrimaryDateColumn)
`PrimaryDateColumn`	Supply a date column if the data is functionally related to it
`IDcols`	Includes PrimaryDateColumn and any other columns you want returned in the validation data with predictions
`ClassWeights`	Utilize these for the classifier model
`SplitRatios`	Supply vector of partition ratios. For example, c(0.70,0.20,0,10).
`task_type`	Set to "GPU" or "CPU"
`ModelID`	Define a character name for your models
`Paths`	A character vector of the path file strings. EITHER SUPPLY 1 file path or N file paths for N models
`SaveModelObjects`	Set to TRUE to save the model objects to file in the folders listed in Paths
`Trees`	Default 15000
`GridTune`	Set to TRUE if you want to grid tune the models
`NumberModelsInGrid`	Set to a numeric value for the number of models to try in grid tune
`NumOfParDepPlots`	Set to pull back N number of partial dependence calibration plots.
`PassInGrid`	Pass in a grid for changing up the parameter settings for catboost

Value

Returns AutoCatBoostRegression() model objects: VariableImportance.csv, Model, ValidationData.csv, EvalutionPlot.png, EvalutionBoxPlot.png, EvaluationMetrics.csv, ParDepPlots.R a named list of features with partial dependence calibration plots, ParDepBoxPlots.R, GridCollect, and catboostgrid

Returns AutoCatBoostRegression() model objects: VariableImportance.csv, Model, ValidationData.csv, EvalutionPlot.png, EvalutionBoxPlot.png, EvaluationMetrics.csv, ParDepPlots.R a named list of features with partial dependence calibration plots, ParDepBoxPlots.R, GridCollect, and catboostgrid

Examples

```
Output <- RemixAutoML::AutoCatBoostdHurdleModel(
  data,
  ValidationData = NULL,
  TestData = NULL,
  Buckets = c(1, 5, 10, 20),
  TargetColumnName = "PLND_LABOR_UNITS",
  FeatureColNames = 4:ncol(data),
  PrimaryDateColumn = "PLND_STRT_DT",
  IDcols = c(1,3),
  ClassWeights = NULL,
  SplitRatios = c(0.7, 0.2, 0.1),
  task_type = "GPU",
  ModelID = "P6",
  Paths = c(paste0(getwd(),"/P6_Buckets")),
  SaveModelObjects = TRUE,
  Trees = 5000,
  GridTune = FALSE,
  MaxModelsInGrid = 1,
  NumOfParDepPlots = 10,
  PassInGrid = grid)
```

AutoCatBoostMultiClass

AutoCatBoostMultiClass is an automated catboost model grid-tuning multinomial classifier and evaluation system

---

Description

AutoCatBoostMultiClass is an automated modeling function that runs a variety of steps. First, a stratified sampling (by the target variable) is done to create train and validation sets. Then, the function will run a random grid tune over N number of models and find which model is the best (a default model is always included in that set). Once the model is identified and built, several other outputs are generated: validation data with predictions, evaluation metrics, variable importance, and column names used in model fitting. You can download the catboost package using devtools, via: devtools::install_github('catboost/catboost', subdir = 'catboost/R-package').

Usage

```
AutoCatBoostMultiClass(data, ValidationData = NULL, TestData = NULL,
  TargetColumnName = NULL, FeatureColNames = NULL,
  PrimaryDateColumn = NULL, ClassWeights = NULL, IDcols = NULL,
  task_type = "GPU", eval_metric = "MultiClassOneVsAll", Trees = 50,
  GridTune = FALSE, grid_eval_metric = "Accuracy",
  MaxModelsInGrid = 10, model_path = NULL, ModelID = "FirstModel",
  ReturnModelObjects = TRUE, SaveModelObjects = FALSE,
  PassInGrid = NULL)
```

Arguments

`data`	This is your data set for training and testing your model
`ValidationData`	This is your holdout data set used in modeling either refine your hyperparameters. Catboost using both training and validation data in the training process so you should evaluate out of sample performance with this data set.
`TestData`	This is your holdout data set. Catboost using both training and validation data in the training process so you should evaluate out of sample performance with this data set.
`TargetColumnName`	Either supply the target column name OR the column number where the target is located, but not mixed types.
`FeatureColNames`	Either supply the feature column names OR the column number where the target is located, but not mixed types. Also, not zero-indexed.
`PrimaryDateColumn`	Supply a date or datetime column for catboost to utilize time as its basis for handling categorical features, instead of random shuffling
`ClassWeights`	Supply a vector of weights for your target classes. E.g. c(0.25, 1) to weight your 0 class by 0.25 and your 1 class by 1.
`IDcols`	A vector of column names or column numbers to keep in your data but not include in the modeling.
`task_type`	"GPU" Set to "GPU" to utilize your GPU for training. Default is "CPU".
`eval_metric`	This is the metric used inside catboost to measure performance on validation data during a grid-tune. "MultiClass" or "MultiClassOneVsAll"
`Trees`	The maximum number of trees you want in your models
`GridTune`	Set to TRUE to run a grid tuning procedure. Set a number in MaxModelsInGrid to tell the procedure how many models you want to test.
`grid_eval_metric`	This is the metric used to find the threshold "auc","accuracy"
`MaxModelsInGrid`	Number of models to test from grid options. 1080 total possible options
`model_path`	A character string of your path file to where you want your output saved
`ModelID`	A character string to name your model and output
`ReturnModelObjects`	Set to TRUE to output all modeling objects. E.g. plots and evaluation metrics
`SaveModelObjects`	Set to TRUE to return all modeling objects to your environment
`PassInGrid`	Defaults to NULL. Pass in a single row of grid from a previous output as a data.table (they are collected as data.tables)

Value

Saves to file and returned in list: VariableImportance.csv, Model (the model), ValidationData.csv, EvaluationMetrics.csv, GridCollect, and GridList

Author(s)

Adrian Antico

See Also

Other Supervised Learning: AutoCatBoostClassifier, AutoCatBoostRegression, AutoCatBoostScoring, AutoH2OMLScoring, AutoH2OModeler, AutoH2OScoring, AutoH2oDRFClassifier, AutoH2oDRFMultiClass, AutoH2oDRFRegression, AutoH2oGBMClassifier, AutoH2oGBMMultiClass, AutoH2oGBMRegression, AutoMLTS, AutoNLS, AutoRecommenderScoring, AutoRecommender, AutoTS, AutoXGBoostClassifier, AutoXGBoostMultiClass, AutoXGBoostRegression, AutoXGBoostScoring

Examples

```
Correl <- 0.85
N <- 1000
data <- data.table::data.table(Target = runif(N))
data[, x1 := qnorm(Target)]
data[, x2 := runif(N)]
data[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable2 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable3 := exp(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                                sqrt(1-Correl^2) * qnorm(x2))))]
data[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                             sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable6 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.10]
data[, Independent_Variable7 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.25]
data[, Independent_Variable8 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.75]
data[, Independent_Variable9 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^2]
data[, Independent_Variable10 := (pnorm(Correl * x1 +
                                          sqrt(1-Correl^2) * qnorm(x2)))^4]
data[, Target := as.factor(
  ifelse(Independent_Variable2 < 0.20, "A",
         ifelse(Independent_Variable2 < 0.40, "B",
                ifelse(Independent_Variable2 < 0.6,  "C",
                       ifelse(Independent_Variable2 < 0.8,  "D", "E")))))]
data[, ':=' (x1 = NULL, x2 = NULL)]
TestModel <- AutoCatBoostMultiClass(data,
                                    ValidationData = NULL,
                                    TestData = NULL,
                                    TargetColumnName = "Target",
                                    FeatureColNames = c(2:11),
                                    PrimaryDateColumn = NULL,
                                    ClassWeights = NULL,
                                    IDcols = NULL,
                                    MaxModelsInGrid = 1,
                                    task_type = "GPU",
                                    eval_metric = "MultiClass",
```

```
                             grid_eval_metric = "Accuracy",
                             Trees = 50,
                             GridTune = FALSE,
                             model_path = NULL,
                             ModelID = "ModelTest",
                             ReturnModelObjects = TRUE,
                             SaveModelObjects = FALSE,
                             PassInGrid = NULL)
```

AutoCatBoostRegression

AutoCatBoostRegression is an automated catboost model grid-tuning classifier and evaluation system

---

Description

AutoCatBoostRegression is an automated modeling function that runs a variety of steps. First, the function will run a random grid tune over N number of models and find which model is the best (a default model is always included in that set). Once the model is identified and built, several other outputs are generated: validation data with predictions, evaluation plot, evaluation boxplot, evaluation metrics, variable importance, partial dependence calibration plots, partial dependence calibration box plots, and column names used in model fitting. You can download the catboost package using devtools, via: devtools::install_github('catboost/catboost', subdir = 'catboost/R-package')

Usage

```
AutoCatBoostRegression(data, ValidationData, TestData = NULL,
  TargetColumnName = NULL, FeatureColNames = NULL,
  PrimaryDateColumn = NULL, IDcols = NULL, task_type = "GPU",
  eval_metric = "RMSE", Alpha = NULL, Trees = 50, GridTune = FALSE,
  grid_eval_metric = "mae", MaxModelsInGrid = 10, model_path = NULL,
  ModelID = "FirstModel", NumOfParDepPlots = 3,
  ReturnModelObjects = TRUE, SaveModelObjects = FALSE,
  PassInGrid = NULL)
```

Arguments

`data` This is your data set for training and testing your model

`ValidationData` This is your holdout data set used in modeling either refine your hyperparameters. Catboost using both training and validation data in the training process so you should evaluate out of sample performance with this data set.

`TestData` This is your holdout data set. Catboost using both training and validation data in the training process so you should evaluate out of sample performance with this data set.

`TargetColumnName`
Either supply the target column name OR the column number where the target is located (but not mixed types).

`FeatureColNames`
: Either supply the feature column names OR the column number where the target is located (but not mixed types)

`PrimaryDateColumn`
: Supply a date or datetime column for catboost to utilize time as its basis for handling categorical features, instead of random shuffling

`IDcols`
: A vector of column names or column numbers to keep in your data but not include in the modeling.

`task_type`
: = "GPU" Set to "GPU" to utilize your GPU for training. Default is "CPU".

`eval_metric`
: This is the metric used inside catboost to measure performance on validation data during a grid-tune. "RMSE" is the default, but other options include: "MAE", "MAPE", "Poisson", "Quantile", "LogLinQuantile", "Lq", "NumErrors", "SMAPE", "R2", "MSLE", "MedianAbsoluteError".

`Alpha`
: This is the quantile value you want to use for quantile regression. Must be a decimal between 0 and 1.

`Trees`
: The maximum number of trees you want in your models

`GridTune`
: Set to TRUE to run a grid tuning procedure. Set a number in MaxModelsInGrid to tell the procedure how many models you want to test.

`grid_eval_metric`
: This is the metric used to find the threshold 'poisson', 'mae', 'mape', 'mse', 'msle', 'kl', 'cs', 'r2'

`MaxModelsInGrid`
: Number of models to test from grid options (1080 total possible options)

`model_path`
: A character string of your path file to where you want your output saved

`ModelID`
: A character string to name your model and output

`NumOfParDepPlots`
: Tell the function the number of partial dependence calibration plots you want to create. Calibration boxplots will only be created for numerical features (not dummy variables)

`ReturnModelObjects`
: Set to TRUE to output all modeling objects (E.g. plots and evaluation metrics)

`SaveModelObjects`
: Set to TRUE to return all modeling objects to your environment

`PassInGrid`
: Defaults to NULL. Pass in a single row of grid from a previous output as a data.table (they are collected as data.tables)

Value

Saves to file and returned in list: VariableImportance.csv, Model, ValidationData.csv, EvalutionPlot.png, EvalutionBoxPlot.png, EvaluationMetrics.csv, ParDepPlots.R a named list of features with partial dependence calibration plots, ParDepBoxPlots.R, GridCollect, and catboostgrid

Author(s)

Adrian Antico

See Also

Other Supervised Learning: AutoCatBoostClassifier, AutoCatBoostMultiClass, AutoCatBoostScoring, AutoH2OMLScoring, AutoH2OModeler, AutoH2OScoring, AutoH2oDRFClassifier, AutoH2oDRFMultiClass, AutoH2oDRFRegression, AutoH2oGBMClassifier, AutoH2oGBMMultiClass, AutoH2oGBMRegression, AutoMLTS, AutoNLS, AutoRecommenderScoring, AutoRecommender, AutoTS, AutoXGBoostClassifier, AutoXGBoostMultiClass, AutoXGBoostRegression, AutoXGBoostScoring

Examples

```
Correl <- 0.85
N <- 1000
data <- data.table::data.table(Target = runif(N))
data[, x1 := qnorm(Target)]
data[, x2 := runif(N)]
data[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable2 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable3 := exp(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                                sqrt(1-Correl^2) * qnorm(x2))))]
data[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                             sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable6 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.10]
data[, Independent_Variable7 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.25]
data[, Independent_Variable8 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.75]
data[, Independent_Variable9 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^2]
data[, Independent_Variable10 := (pnorm(Correl * x1 +
                                          sqrt(1-Correl^2) * qnorm(x2)))^4]
data[, Independent_Variable11 := as.factor(
  ifelse(Independent_Variable2 < 0.20, "A",
         ifelse(Independent_Variable2 < 0.40, "B",
                ifelse(Independent_Variable2 < 0.6,  "C",
                       ifelse(Independent_Variable2 < 0.8,  "D", "E")))))]
data[, ':=' (x1 = NULL, x2 = NULL)]
TestModel <- AutoCatBoostRegression(data,
                                    ValidationData = NULL,
                                    TestData = NULL,
                                    TargetColumnName = "Target",
                                    FeatureColNames = c(2:12),
                                    PrimaryDateColumn = NULL,
                                    IDcols = NULL,
                                    MaxModelsInGrid = 1,
                                    task_type = "GPU",
                                    eval_metric = "RMSE",
                                    grid_eval_metric = "r2",
                                    Trees = 50,
                                    GridTune = FALSE,
                                    model_path = NULL,
                                    ModelID = "ModelTest",
```

```
                                    NumOfParDepPlots = 3,
                                    ReturnModelObjects = TRUE,
                                    SaveModelObjects = FALSE,
                                    PassInGrid = NULL)
```

`AutoCatBoostScoring` *AutoCatBoostScoring is an automated scoring function that compliments the AutoCatBoost model training functions.*

Description

AutoCatBoostScoring is an automated scoring function that compliments the AutoCatBoost model training functions. This function requires you to supply features for scoring. It will run ModelDataPrep() to prepare your features for catboost data conversion and scoring.

Usage

```
AutoCatBoostScoring(TargetType = NULL, ScoringData = NULL,
  FeatureColumnNames = NULL, IDcols = NULL, ModelObject = NULL,
  ModelPath = NULL, ModelID = NULL, ReturnFeatures = TRUE,
  MDP_Impute = TRUE, MDP_CharToFactor = TRUE, MDP_RemoveDates = TRUE,
  MDP_MissFactor = "0", MDP_MissNum = -1)
```

Arguments

`TargetType` Set this value to "regression", "classification", or "multiclass" to score models built using AutoCatBoostRegression(), AutoCatBoostClassify() or AutoCatBoostMultiClass().

`ScoringData` This is your data.table of features for scoring. Can be a single row or batch.

`FeatureColumnNames` Supply either column names or column numbers used in the AutoCatBoostRegression() function

`IDcols` Supply ID column numbers for any metadata you want returned with your predicted values

`ModelObject` Supply the model object directly for scoring instead of loading it from file. If you supply this, ModelID and ModelPath will be ignored.

`ModelPath` Supply your path file used in the AutoCatBoost__() function

`ModelID` Supply the model ID used in the AutoCatBoost__() function

`ReturnFeatures` Set to TRUE to return your features with the predicted values.

`MDP_Impute` Set to TRUE if you did so for modeling and didn't do so before supplying ScoringData in this function

`MDP_CharToFactor` Set to TRUE to turn your character columns to factors if you didn't do so to your ScoringData that you are supplying to this function

`MDP_RemoveDates` Set to TRUE if you have date of timestamp columns in your ScoringData

`MDP_MissFactor` If you set MDP_Impute to TRUE, supply the character values to replace missing values with

`MDP_MissNum` If you set MDP_Impute to TRUE, supply a numeric value to replace missing values with

Value

A data.table of predicted values with the option to return model features as well.

See Also

Other Supervised Learning: AutoCatBoostClassifier, AutoCatBoostMultiClass, AutoCatBoostRegression, AutoH2OMLScoring, AutoH2OModeler, AutoH2OScoring, AutoH2oDRFClassifier, AutoH2oDRFMultiClass, AutoH2oDRFRegression, AutoH2oGBMClassifier, AutoH2oGBMMultiClass, AutoH2oGBMRegression, AutoMLTS, AutoNLS, AutoRecommenderScoring, AutoRecommender, AutoTS, AutoXGBoostClassifier, AutoXGBoostMultiClass, AutoXGBoostRegression, AutoXGBoostScoring

Examples

```
Preds <- AutoCatBoostScoring(TargetType = "regression",
                             ScoringData = data,
                             FeatureColumnNames = 2:12,
                             IDcols = NULL,
                             ModelObject = NULL,
                             ModelPath = "home",
                             ModelID = "ModelTest",
                             ReturnFeatures = TRUE,
                             MDP_Impute = TRUE,
                             MDP_CharToFactor = TRUE,
                             MDP_RemoveDates = TRUE,
                             MDP_MissFactor = "0",
                             MDP_MissNum = -1)
```

AutoDataPartition *The AutoDataPartition function*

Description

This function will take your ratings matrix and model and score your data in parallel.

Usage

```
AutoDataPartition(data, NumDataSets = 3, Ratios = c(0.7, 0.2, 0.1),
  PartitionType = "random", StratifyColumnNames = NULL,
  TimeColumnName = NULL)
```

Arguments

`data` Source data to do your partitioning on

`NumDataSets` The number of total data sets you want built

`Ratios` A vector of values for how much data each data set should get in each split. E.g. c(0.70, 0.20, 0.10)

`PartitionType` Set to either "random", "timeseries", or "time". With "random", your data will be paritioned randomly (with stratified sampling if column names are supplied). With "timeseries", you can partition by time with a stratify option (so long as you have an equal number of records for each strata). With "time" you will have data sets generated so that the training data contains the earliest records in time, validation data the second earliest, test data the third earliest, etc.

`StratifyColumnNames` Supply column names of categorical features to use in a stratified sampling procedure for partitioning the data. Partition type must be "random" to use this option

`TimeColumnName` Supply a date column name or a name of a column with an ID for sorting by time such that the smallest number is the earliest in time.

Value

Returns a list of data.tables

Author(s)

Adrian Antico and Douglas Pestana

See Also

Other Feature Engineering: AutoWord2VecModeler, CreateCalendarVariables, DT_GDL_Feature_Engineering, DummifyDT, FAST_GDL_Feature_Engineering, GDL_Feature_Engineering, ModelDataPrep, Scoring_GDL_Feature_E

Examples

```
dataSets <- AutoDataPartition(data,
                              NumDataSets = 3,
                              Ratios = c(0.70,0.20,0.10),
                              PartitionType = "random",
                              StratifyColumnNames = NULL,
                              TimeColumnName = NULL)
```

`AutoH2oDRFClassifier` *AutoH2oDRFClassifier is an automated H2O modeling framework with grid-tuning and model evaluation*

Description

AutoH2oDRFClassifier is an automated H2O modeling framework with grid-tuning and model evaluation that runs a variety of steps. First, a stratified sampling (by the target variable) is done to create train and validation sets. Then, the function will run a random grid tune over N number of models and find which model is the best (a default model is always included in that set). Once the model is identified and built, several other outputs are generated: validation data with predictions, evaluation plot, evaluation metrics, variable importance, partial dependence calibration plots, and column names used in model fitting.

Usage

```
AutoH2oDRFClassifier(data, ValidationData = NULL, TestData = NULL,
  TargetColumnName = NULL, FeatureColNames = NULL,
  eval_metric = "auc", Trees = 50, GridTune = FALSE,
  MaxMem = "32G", MaxModelsInGrid = 2, model_path = NULL,
  ModelID = "FirstModel", NumOfParDepPlots = 3,
  ReturnModelObjects = TRUE, SaveModelObjects = FALSE,
  IfSaveModel = "mojo")
```

Arguments

`data`	This is your data set for training and testing your model
`ValidationData`	This is your holdout data set used in modeling either refine your hyperparameters.
`TestData`	This is your holdout data set. Catboost using both training and validation data in the training process so you should evaluate out of sample performance with this data set.
`TargetColumnName`	Either supply the target column name OR the column number where the target is located (but not mixed types). Note that the target column needs to be a 0 — 1 numeric variable.
`FeatureColNames`	Either supply the feature column names OR the column number where the target is located (but not mixed types)
`eval_metric`	This is the metric used to identify best grid tuned model. Choose from "AUC" or "logloss"
`Trees`	The maximum number of trees you want in your models
`GridTune`	Set to TRUE to run a grid tuning procedure. Set a number in MaxModelsInGrid to tell the procedure how many models you want to test.
`MaxMem`	Set the maximum amount of memory you'd like to dedicate to the model run. E.g. "32G"
`MaxModelsInGrid`	Number of models to test from grid options (1080 total possible options)
`model_path`	A character string of your path file to where you want your output saved
`ModelID`	A character string to name your model and output
`NumOfParDepPlots`	Tell the function the number of partial dependence calibration plots you want to create.

ReturnModelObjects
: Set to TRUE to output all modeling objects (E.g. plots and evaluation metrics)

SaveModelObjects
: Set to TRUE to return all modeling objects to your environment

IfSaveModel
: Set to "mojo" to save a mojo file, otherwise "standard" to save a regular H2O model object

Value

Saves to file and returned in list: VariableImportance.csv, Model, ValidationData.csv, EvalutionPlot.png, EvaluationMetrics.csv, ParDepPlots.R a named list of features with partial dependence calibration plots, GridCollect, and GridList

Author(s)

Adrian Antico

See Also

Other Supervised Learning: AutoCatBoostClassifier, AutoCatBoostMultiClass, AutoCatBoostRegression, AutoCatBoostScoring, AutoH2OMLScoring, AutoH2OModeler, AutoH2OScoring, AutoH2oDRFMultiClass, AutoH2oDRFRegression, AutoH2oGBMClassifier, AutoH2oGBMMultiClass, AutoH2oGBMRegression, AutoMLTS, AutoNLS, AutoRecommenderScoring, AutoRecommender, AutoTS, AutoXGBoostClassifier, AutoXGBoostMultiClass, AutoXGBoostRegression, AutoXGBoostScoring

Examples

```
Correl <- 0.85
N <- 1000
data <- data.table::data.table(Target = runif(N))
data[, x1 := qnorm(Target)]
data[, x2 := runif(N)]
data[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable2 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable3 := exp(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                                sqrt(1-Correl^2) * qnorm(x2))))]
data[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                             sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable6 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.10]
data[, Independent_Variable7 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.25]
data[, Independent_Variable8 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.75]
data[, Independent_Variable9 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^2]
data[, Independent_Variable10 := (pnorm(Correl * x1 +
                                          sqrt(1-Correl^2) * qnorm(x2)))^4]
data[, Independent_Variable11 := as.factor(
  ifelse(Independent_Variable2 < 0.20, "A",
```

```
        ifelse(Independent_Variable2 < 0.40, "B",
               ifelse(Independent_Variable2 < 0.6,  "C",
                      ifelse(Independent_Variable2 < 0.8,  "D", "E")))))]
data[, ':=' (x1 = NULL, x2 = NULL)]
data[, Target := as.factor(ifelse(Independent_Variable2 < 0.5, 1, 0))]
TestModel <- AutoH2oDRFClassifier(data,
                                  ValidationData = NULL,
                                  TestData = NULL,
                                  TargetColumnName = "Target",
                                  FeatureColNames = 2:ncol(data),
                                  eval_metric = "auc",
                                  Trees = 50,
                                  GridTune = FALSE,
                                  MaxMem = "32G",
                                  MaxModelsInGrid = 10,
                                  model_path = NULL,
                                  ModelID = "FirstModel",
                                  NumOfParDepPlots = 3,
                                  ReturnModelObjects = TRUE,
                                  SaveModelObjects = FALSE,
                                  IfSaveModel = "mojo")
```

`AutoH2oDRFMultiClass` *AutoH2oDRFMultiClass is an automated H2O modeling framework with grid-tuning and model evaluation*

Description

AutoH2oDRFMultiClass is an automated H2O modeling framework with grid-tuning and model evaluation that runs a variety of steps. First, a stratified sampling (by the target variable) is done to create train and validation sets. Then, the function will run a random grid tune over N number of models and find which model is the best (a default model is always included in that set). Once the model is identified and built, several other outputs are generated: validation data with predictions, evaluation metrics, confusion matrix, and variable importance.

Usage

```
AutoH2oDRFMultiClass(data, ValidationData = NULL, TestData = NULL,
  TargetColumnName = NULL, FeatureColNames = NULL,
  eval_metric = "logloss", Trees = 50, GridTune = FALSE,
  MaxMem = "32G", MaxModelsInGrid = 2, model_path = NULL,
  ModelID = "FirstModel", ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE, IfSaveModel = "mojo")
```

Arguments

`data`	This is your data set for training and testing your model
`ValidationData`	This is your holdout data set used in modeling either refine your hyperparameters.

`TestData` This is your holdout data set. Catboost using both training and validation data in the training process so you should evaluate out of sample performance with this data set.

`TargetColumnName` Either supply the target column name OR the column number where the target is located (but not mixed types).

`FeatureColNames` Either supply the feature column names OR the column number where the target is located (but not mixed types)

`eval_metric` This is the metric used to identify best grid tuned model. Choose from "logloss", "r2", "RMSE", "MSE"

`Trees` The maximum number of trees you want in your models

`GridTune` Set to TRUE to run a grid tuning procedure. Set a number in MaxModelsInGrid to tell the procedure how many models you want to test.

`MaxMem` Set the maximum amount of memory you'd like to dedicate to the model run. E.g. "32G"

`MaxModelsInGrid` Number of models to test from grid options (1080 total possible options)

`model_path` A character string of your path file to where you want your output saved

`ModelID` A character string to name your model and output

`ReturnModelObjects` Set to TRUE to output all modeling objects (E.g. plots and evaluation metrics)

`SaveModelObjects` Set to TRUE to return all modeling objects to your environment

`IfSaveModel` Set to "mojo" to save a mojo file, otherwise "standard" to save a regular H2O model object

Value

Saves to file and returned in list: VariableImportance.csv, Model, ValidationData.csv, EvaluationMetrics.csv, GridCollect, and GridList

Author(s)

Adrian Antico

See Also

Other Supervised Learning: `AutoCatBoostClassifier`, `AutoCatBoostMultiClass`, `AutoCatBoostRegression`, `AutoCatBoostScoring`, `AutoH2OMLScoring`, `AutoH2OModeler`, `AutoH2OScoring`, `AutoH2oDRFClassifier`, `AutoH2oDRFRegression`, `AutoH2oGBMClassifier`, `AutoH2oGBMMultiClass`, `AutoH2oGBMRegression`, `AutoMLTS`, `AutoNLS`, `AutoRecommenderScoring`, `AutoRecommender`, `AutoTS`, `AutoXGBoostClassifier`, `AutoXGBoostMultiClass`, `AutoXGBoostRegression`, `AutoXGBoostScoring`

Examples

```
Correl <- 0.85
N <- 1000
data <- data.table::data.table(Target = runif(N))
```

```
data[, x1 := qnorm(Target)]
data[, x2 := runif(N)]
data[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                        sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable2 := (pnorm(Correl * x1 +
                                     sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable3 := exp(pnorm(Correl * x1 +
                                        sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2))))]
data[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable6 := (pnorm(Correl * x1 +
                                     sqrt(1-Correl^2) * qnorm(x2)))^0.10]
data[, Independent_Variable7 := (pnorm(Correl * x1 +
                                     sqrt(1-Correl^2) * qnorm(x2)))^0.25]
data[, Independent_Variable8 := (pnorm(Correl * x1 +
                                     sqrt(1-Correl^2) * qnorm(x2)))^0.75]
data[, Independent_Variable9 := (pnorm(Correl * x1 +
                                     sqrt(1-Correl^2) * qnorm(x2)))^2]
data[, Independent_Variable10 := (pnorm(Correl * x1 +
                                      sqrt(1-Correl^2) * qnorm(x2)))^4]
data[, Independent_Variable11 := as.factor(
  ifelse(Independent_Variable2 < 0.20, "A",
         ifelse(Independent_Variable2 < 0.40, "B",
                ifelse(Independent_Variable2 < 0.6,  "C",
                       ifelse(Independent_Variable2 < 0.8,  "D", "E")))))]
data[, Target :=
ifelse(Independent_Variable2 < 0.25, "A",
       ifelse(Independent_Variable2 < 0.45, "B",
              ifelse(Independent_Variable2 < 0.65, "C",
                     ifelse(Independent_Variable2 < 0.85,  "D", "E"))))]
data[, ':=' (x1 = NULL, x2 = NULL)]
TestModel <- AutoH2oDRFMultiClass(data,
                                  ValidationData = NULL,
                                  TestData = NULL,
                                  TargetColumnName = "Target",
                                  FeatureColNames = 2:ncol(data),
                                  eval_metric = "logloss",
                                  Trees = 50,
                                  GridTune = FALSE,
                                  MaxMem = "32G",
                                  MaxModelsInGrid = 10,
                                  model_path = NULL,
                                  ModelID = "FirstModel",
                                  ReturnModelObjects = TRUE,
                                  SaveModelObjects = FALSE,
                                  IfSaveModel = "mojo")
```

`AutoH2oDRFRegression` *AutoH2oDRFRegression is an automated H2O modeling framework with grid-tuning and model evaluation*

Description

AutoH2oDRFRegression is an automated H2O modeling framework with grid-tuning and model evaluation that runs a variety of steps. First, the function will run a random grid tune over N number of models and find which model is the best (a default model is always included in that set). Once the model is identified and built, several other outputs are generated: validation data with predictions, evaluation plot, evaluation boxplot, evaluation metrics, variable importance, partial dependence calibration plots, partial dependence calibration box plots, and column names used in model fitting.

Usage

```
AutoH2oDRFRegression(data, ValidationData = NULL, TestData = NULL,
  TargetColumnName = NULL, FeatureColNames = NULL,
  eval_metric = "RMSE", Trees = 50, GridTune = FALSE,
  MaxMem = "32G", MaxModelsInGrid = 2, model_path = NULL,
  ModelID = "FirstModel", NumOfParDepPlots = 3,
  ReturnModelObjects = TRUE, SaveModelObjects = FALSE,
  IfSaveModel = "mojo")
```

Arguments

`data`	This is your data set for training and testing your model
`ValidationData`	This is your holdout data set used in modeling either refine your hyperparameters.
`TestData`	This is your holdout data set. Catboost using both training and validation data in the training process so you should evaluate out of sample performance with this data set.
`TargetColumnName`	Either supply the target column name OR the column number where the target is located (but not mixed types).
`FeatureColNames`	Either supply the feature column names OR the column number where the target is located (but not mixed types)
`eval_metric`	This is the metric used to identify best grid tuned model. Choose from "MSE", "RMSE", "MAE", "RMSLE"
`Trees`	The maximum number of trees you want in your models
`GridTune`	Set to TRUE to run a grid tuning procedure. Set a number in MaxModelsInGrid to tell the procedure how many models you want to test.
`MaxMem`	Set the maximum amount of memory you'd like to dedicate to the model run. E.g. "32G"
`MaxModelsInGrid`	Number of models to test from grid options (1080 total possible options)
`model_path`	A character string of your path file to where you want your output saved
`ModelID`	A character string to name your model and output
`NumOfParDepPlots`	Tell the function the number of partial dependence calibration plots you want to create. Calibration boxplots will only be created for numerical features (not dummy variables)

ReturnModelObjects
: Set to TRUE to output all modeling objects (E.g. plots and evaluation metrics)

SaveModelObjects
: Set to TRUE to return all modeling objects to your environment

IfSaveModel
: Set to "mojo" to save a mojo file, otherwise "standard" to save a regular H2O model object

Value

Saves to file and returned in list: VariableImportance.csv, Model, ValidationData.csv, EvalutionPlot.png, EvalutionBoxPlot.png, EvaluationMetrics.csv, ParDepPlots.R a named list of features with partial dependence calibration plots, ParDepBoxPlots.R, GridCollect, and GridList

Author(s)

Adrian Antico

See Also

Other Supervised Learning: AutoCatBoostClassifier, AutoCatBoostMultiClass, AutoCatBoostRegression, AutoCatBoostScoring, AutoH2OMLScoring, AutoH2OModeler, AutoH2OScoring, AutoH2oDRFClassifier, AutoH2oDRFMultiClass, AutoH2oGBMClassifier, AutoH2oGBMMultiClass, AutoH2oGBMRegression, AutoMLTS, AutoNLS, AutoRecommenderScoring, AutoRecommender, AutoTS, AutoXGBoostClassifier, AutoXGBoostMultiClass, AutoXGBoostRegression, AutoXGBoostScoring

Examples

```
Correl <- 0.85
N <- 1000
data <- data.table::data.table(Target = runif(N))
data[, x1 := qnorm(Target)]
data[, x2 := runif(N)]
data[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable2 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable3 := exp(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                                sqrt(1-Correl^2) * qnorm(x2))))]
data[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                             sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable6 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.10]
data[, Independent_Variable7 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.25]
data[, Independent_Variable8 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.75]
data[, Independent_Variable9 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^2]
data[, Independent_Variable10 := (pnorm(Correl * x1 +
                                          sqrt(1-Correl^2) * qnorm(x2)))^4]
data[, Independent_Variable11 := as.factor(
```

```
  ifelse(Independent_Variable2 < 0.20, "A",
         ifelse(Independent_Variable2 < 0.40, "B",
                ifelse(Independent_Variable2 < 0.6,  "C",
                       ifelse(Independent_Variable2 < 0.8,  "D", "E")))))]
data[, ':=' (x1 = NULL, x2 = NULL)]
TestModel <- AutoH2oDRFRegression(data,
                                  ValidationData = NULL,
                                  TestData = NULL,
                                  TargetColumnName = "Target",
                                  FeatureColNames = 2:ncol(data),
                                  eval_metric = "RMSE",
                                  Trees = 50,
                                  GridTune = FALSE,
                                  MaxMem = "32G",
                                  MaxModelsInGrid = 10,
                                  model_path = NULL,
                                  ModelID = "FirstModel",
                                  NumOfParDepPlots = 3,
                                  ReturnModelObjects = TRUE,
                                  SaveModelObjects = FALSE,
                                  IfSaveModel = "mojo")
```

`AutoH2oGBMClassifier` *AutoH2oGBMClassifier is an automated H2O modeling framework with grid-tuning and model evaluation*

Description

AutoH2oGBMClassifier is an automated H2O modeling framework with grid-tuning and model evaluation that runs a variety of steps. First, a stratified sampling (by the target variable) is done to create train and validation sets. Then, the function will run a random grid tune over N number of models and find which model is the best (a default model is always included in that set). Once the model is identified and built, several other outputs are generated: validation data with predictions, evaluation plot, evaluation metrics, variable importance, partial dependence calibration plots, and column names used in model fitting.

Usage

```
AutoH2oGBMClassifier(data, ValidationData = NULL, TestData = NULL,
  TargetColumnName = NULL, FeatureColNames = NULL,
  eval_metric = "auc", Trees = 50, GridTune = FALSE,
  MaxMem = "32G", MaxModelsInGrid = 2, model_path = NULL,
  ModelID = "FirstModel", NumOfParDepPlots = 3,
  ReturnModelObjects = TRUE, SaveModelObjects = FALSE,
  IfSaveModel = "mojo")
```

Arguments

`data`	This is your data set for training and testing your model
`ValidationData`	This is your holdout data set used in modeling either refine your hyperparameters.

`TestData` This is your holdout data set. Catboost using both training and validation data in the training process so you should evaluate out of sample performance with this data set.

`TargetColumnName` Either supply the target column name OR the column number where the target is located (but not mixed types). Note that the target column needs to be a 0 — 1 numeric variable.

`FeatureColNames` Either supply the feature column names OR the column number where the target is located (but not mixed types)

`eval_metric` This is the metric used to identify best grid tuned model. Choose from "AUC" or "logloss"

`Trees` The maximum number of trees you want in your models

`GridTune` Set to TRUE to run a grid tuning procedure. Set a number in MaxModelsInGrid to tell the procedure how many models you want to test.

`MaxMem` Set the maximum amount of memory you'd like to dedicate to the model run. E.g. "32G"

`MaxModelsInGrid` Number of models to test from grid options (1080 total possible options)

`model_path` A character string of your path file to where you want your output saved

`ModelID` A character string to name your model and output

`NumOfParDepPlots` Tell the function the number of partial dependence calibration plots you want to create.

`ReturnModelObjects` Set to TRUE to output all modeling objects (E.g. plots and evaluation metrics)

`SaveModelObjects` Set to TRUE to return all modeling objects to your environment

`IfSaveModel` Set to "mojo" to save a mojo file, otherwise "standard" to save a regular H2O model object

Value

Saves to file and returned in list: VariableImportance.csv, Model, ValidationData.csv, EvalutionPlot.png, EvaluationMetrics.csv, ParDepPlots.R a named list of features with partial dependence calibration plots, GridCollect, and GridList

Author(s)

Adrian Antico

See Also

Other Supervised Learning: `AutoCatBoostClassifier`, `AutoCatBoostMultiClass`, `AutoCatBoostRegression`, `AutoCatBoostScoring`, `AutoH2OMLScoring`, `AutoH2OModeler`, `AutoH2OScoring`, `AutoH2oDRFClassifier`, `AutoH2oDRFMultiClass`, `AutoH2oDRFRegression`, `AutoH2oGBMMultiClass`, `AutoH2oGBMRegression`, `AutoMLTS`, `AutoNLS`, `AutoRecommenderScoring`, `AutoRecommender`, `AutoTS`, `AutoXGBoostClassifier`, `AutoXGBoostMultiClass`, `AutoXGBoostRegression`, `AutoXGBoostScoring`

Examples

```
Correl <- 0.85
N <- 1000
data <- data.table::data.table(Target = runif(N))
data[, x1 := qnorm(Target)]
data[, x2 := runif(N)]
data[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable2 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable3 := exp(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                                sqrt(1-Correl^2) * qnorm(x2))))]
data[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                             sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable6 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.10]
data[, Independent_Variable7 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.25]
data[, Independent_Variable8 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.75]
data[, Independent_Variable9 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^2]
data[, Independent_Variable10 := (pnorm(Correl * x1 +
                                          sqrt(1-Correl^2) * qnorm(x2)))^4]
data[, Independent_Variable11 := as.factor(
  ifelse(Independent_Variable2 < 0.20, "A",
         ifelse(Independent_Variable2 < 0.40, "B",
                ifelse(Independent_Variable2 < 0.6,  "C",
                       ifelse(Independent_Variable2 < 0.8,  "D", "E")))))]
data[, ':=' (x1 = NULL, x2 = NULL)]
data[, Target := as.factor(ifelse(Independent_Variable2 < 0.5, 1, 0))]
TestModel <- AutoH2oGBMClassifier(data,
                                  ValidationData = NULL,
                                  TestData = NULL,
                                  TargetColumnName = "Target",
                                  FeatureColNames = 2:ncol(data),
                                  eval_metric = "auc",
                                  Trees = 50,
                                  GridTune = FALSE,
                                  MaxMem = "32G",
                                  MaxModelsInGrid = 10,
                                  model_path = NULL,
                                  ModelID = "FirstModel",
                                  NumOfParDepPlots = 3,
                                  ReturnModelObjects = TRUE,
                                  SaveModelObjects = FALSE,
                                  IfSaveModel = "mojo")
```

AutoH2oGBMMultiClass *AutoH2oGBMMultiClass is an automated H2O modeling framework with grid-tuning and model evaluation*

Description

AutoH2oGBMMultiClass is an automated H2O modeling framework with grid-tuning and model evaluation that runs a variety of steps. First, a stratified sampling (by the target variable) is done to create train and validation sets. Then, the function will run a random grid tune over N number of models and find which model is the best (a default model is always included in that set). Once the model is identified and built, several other outputs are generated: validation data with predictions, evaluation metrics, confusion matrix, and variable importance.

Usage

```
AutoH2oGBMMultiClass(data, ValidationData = NULL, TestData = NULL,
  TargetColumnName = NULL, FeatureColNames = NULL,
  eval_metric = "logloss", Trees = 50, GridTune = FALSE,
  MaxMem = "32G", MaxModelsInGrid = 2, model_path = NULL,
  ModelID = "FirstModel", ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE, IfSaveModel = "mojo")
```

Arguments

`data`	This is your data set for training and testing your model
`ValidationData`	This is your holdout data set used in modeling either refine your hyperparameters.
`TestData`	This is your holdout data set. Catboost using both training and validation data in the training process so you should evaluate out of sample performance with this data set.
`TargetColumnName`	Either supply the target column name OR the column number where the target is located (but not mixed types).
`FeatureColNames`	Either supply the feature column names OR the column number where the target is located (but not mixed types)
`eval_metric`	This is the metric used to identify best grid tuned model. Choose from "logloss", "r2", "RMSE", "MSE"
`Trees`	The maximum number of trees you want in your models
`GridTune`	Set to TRUE to run a grid tuning procedure. Set a number in MaxModelsInGrid to tell the procedure how many models you want to test.
`MaxMem`	Set the maximum amount of memory you'd like to dedicate to the model run. E.g. "32G"
`MaxModelsInGrid`	Number of models to test from grid options (1080 total possible options)
`model_path`	A character string of your path file to where you want your output saved
`ModelID`	A character string to name your model and output
`ReturnModelObjects`	Set to TRUE to output all modeling objects (E.g. plots and evaluation metrics)
`SaveModelObjects`	Set to TRUE to return all modeling objects to your environment
`IfSaveModel`	Set to "mojo" to save a mojo file, otherwise "standard" to save a regular H2O model object

Value

Saves to file and returned in list: VariableImportance.csv, Model, ValidationData.csv, EvaluationMetrics.csv, GridCollect, and GridList

Author(s)

Adrian Antico

See Also

Other Supervised Learning: AutoCatBoostClassifier, AutoCatBoostMultiClass, AutoCatBoostRegression, AutoCatBoostScoring, AutoH2OMLScoring, AutoH2OModeler, AutoH2OScoring, AutoH2oDRFClassifier, AutoH2oDRFMultiClass, AutoH2oDRFRegression, AutoH2oGBMClassifier, AutoH2oGBMRegression, AutoMLTS, AutoNLS, AutoRecommenderScoring, AutoRecommender, AutoTS, AutoXGBoostClassifier, AutoXGBoostMultiClass, AutoXGBoostRegression, AutoXGBoostScoring

Examples

```
Correl <- 0.85
N <- 1000
data <- data.table::data.table(Target = runif(N))
data[, x1 := qnorm(Target)]
data[, x2 := runif(N)]
data[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable2 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable3 := exp(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                                sqrt(1-Correl^2) * qnorm(x2))))]
data[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                             sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable6 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.10]
data[, Independent_Variable7 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.25]
data[, Independent_Variable8 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.75]
data[, Independent_Variable9 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^2]
data[, Independent_Variable10 := (pnorm(Correl * x1 +
                                          sqrt(1-Correl^2) * qnorm(x2)))^4]
data[, Independent_Variable11 := as.factor(
  ifelse(Independent_Variable2 < 0.20, "A",
         ifelse(Independent_Variable2 < 0.40, "B",
                ifelse(Independent_Variable2 < 0.6,  "C",
                       ifelse(Independent_Variable2 < 0.8,  "D", "E")))))]
data[, Target :=
ifelse(Independent_Variable2 < 0.25, "A",
       ifelse(Independent_Variable2 < 0.45, "B",
              ifelse(Independent_Variable2 < 0.65, "C",
                     ifelse(Independent_Variable2 < 0.85,  "D", "E"))))]
data[, ':=' (x1 = NULL, x2 = NULL)]
TestModel <- AutoH2oGBMMultiClass(data,
```

```
                    ValidationData = NULL,
                    TestData = NULL,
                    TargetColumnName = "Target",
                    FeatureColNames = 2:ncol(data),
                    eval_metric = "logloss",
                    Trees = 50,
                    GridTune = FALSE,
                    MaxMem = "32G",
                    MaxModelsInGrid = 10,
                    model_path = NULL,
                    ModelID = "FirstModel",
                    ReturnModelObjects = TRUE,
                    SaveModelObjects = FALSE,
                    IfSaveModel = "mojo")
```

`AutoH2oGBMRegression` *AutoH2oGBMRegression is an automated H2O modeling framework with grid-tuning and model evaluation*

Description

AutoH2oGBMRegression is an automated H2O modeling framework with grid-tuning and model evaluation that runs a variety of steps. First, the function will run a random grid tune over N number of models and find which model is the best (a default model is always included in that set). Once the model is identified and built, several other outputs are generated: validation data with predictions, evaluation plot, evaluation boxplot, evaluation metrics, variable importance, partial dependence calibration plots, partial dependence calibration box plots, and column names used in model fitting.

Usage

```
AutoH2oGBMRegression(data, ValidationData, TestData = NULL,
  TargetColumnName = NULL, FeatureColNames = NULL, Alpha = NULL,
  Distribution = "poisson", eval_metric = "RMSE",
  TrainSplitRatio = 0.8, Trees = 50, GridTune = FALSE,
  MaxMem = "32G", MaxModelsInGrid = 2, model_path = NULL,
  ModelID = "FirstModel", NumOfParDepPlots = 3,
  ReturnModelObjects = TRUE, SaveModelObjects = FALSE,
  IfSaveModel = "mojo")
```

Arguments

`data` This is your data set for training and testing your model

`ValidationData` This is your holdout data set used in modeling either refine your hyperparameters.

`TestData` This is your holdout data set. Catboost using both training and validation data in the training process so you should evaluate out of sample performance with this data set.

`TargetColumnName`
Either supply the target column name OR the column number where the target is located (but not mixed types).

FeatureColNames
: Either supply the feature column names OR the column number where the target is located (but not mixed types)

Alpha
: This is the quantile value you want to use for quantile regression. Must be a decimal between 0 and 1.

Distribution
: Choose from gaussian", "poisson", "gamma", "tweedie", "laplace", "quantile", "huber"

eval_metric
: This is the metric used to identify best grid tuned model. Choose from "MSE", "RMSE", "MAE", "RMSLE"

TrainSplitRatio
: A decimal between 0.01 and 0.99 that tells the function how much data to keep for training and validation.

Trees
: The maximum number of trees you want in your models

GridTune
: Set to TRUE to run a grid tuning procedure. Set a number in MaxModelsInGrid to tell the procedure how many models you want to test.

MaxMem
: Set the maximum amount of memory you'd like to dedicate to the model run. E.g. "32G"

MaxModelsInGrid
: Number of models to test from grid options (1080 total possible options)

model_path
: A character string of your path file to where you want your output saved

ModelID
: A character string to name your model and output

NumOfParDepPlots
: Tell the function the number of partial dependence calibration plots you want to create. Calibration boxplots will only be created for numerical features (not dummy variables)

ReturnModelObjects
: Set to TRUE to output all modeling objects (E.g. plots and evaluation metrics)

SaveModelObjects
: Set to TRUE to return all modeling objects to your environment

IfSaveModel
: Set to "mojo" to save a mojo file, otherwise "standard" to save a regular H2O model object

Value

Saves to file and returned in list: VariableImportance.csv, Model, ValidationData.csv, EvalutionPlot.png, EvalutionBoxPlot.png, EvaluationMetrics.csv, ParDepPlots.R a named list of features with partial dependence calibration plots, ParDepBoxPlots.R, GridCollect, and GridList

Author(s)

Adrian Antico

See Also

Other Supervised Learning: AutoCatBoostClassifier, AutoCatBoostMultiClass, AutoCatBoostRegression, AutoCatBoostScoring, AutoH2OMLScoring, AutoH2OModeler, AutoH2OScoring, AutoH2oDRFClassifier, AutoH2oDRFMultiClass, AutoH2oDRFRegression, AutoH2oGBMClassifier, AutoH2oGBMMultiClass, AutoMLTS, AutoNLS, AutoRecommenderScoring, AutoRecommender, AutoTS, AutoXGBoostClassifier, AutoXGBoostMultiClass, AutoXGBoostRegression, AutoXGBoostScoring

Examples

```
Correl <- 0.85
N <- 1000
data <- data.table::data.table(Target = runif(N))
data[, x1 := qnorm(Target)]
data[, x2 := runif(N)]
data[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable2 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable3 := exp(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                                sqrt(1-Correl^2) * qnorm(x2))))]
data[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                             sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable6 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.10]
data[, Independent_Variable7 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.25]
data[, Independent_Variable8 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.75]
data[, Independent_Variable9 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^2]
data[, Independent_Variable10 := (pnorm(Correl * x1 +
                                          sqrt(1-Correl^2) * qnorm(x2)))^4]
data[, Independent_Variable11 := as.factor(
  ifelse(Independent_Variable2 < 0.20, "A",
         ifelse(Independent_Variable2 < 0.40, "B",
                ifelse(Independent_Variable2 < 0.6,  "C",
                       ifelse(Independent_Variable2 < 0.8,  "D", "E")))))]
data[, ':=' (x1 = NULL, x2 = NULL)]
TestModel <- AutoH2oGBMRegression(data,
                                  ValidationData = NULL,
                                  TestData = NULL,
                                  TargetColumnName = "Target",
                                  FeatureColNames = 2:ncol(data),
                                  Alpha = NULL,
                                  Distribution = "poisson",
                                  eval_metric = "RMSE",
                                  Trees = 50,
                                  GridTune = FALSE,
                                  MaxMem = "32G",
                                  MaxModelsInGrid = 10,
                                  model_path = NULL,
                                  ModelID = "FirstModel",
                                  NumOfParDepPlots = 3,
                                  ReturnModelObjects = TRUE,
                                  SaveModelObjects = FALSE,
                                  IfSaveModel = "mojo")
```

AutoH2OMLScoring — *AutoH2OMLScoring is an automated scoring function that compliments the AutoH2o model training functions.*

Description

AutoH2OMLScoring is an automated scoring function that compliments the AutoH2oGBM__() and AutoH2oDRF__() models training functions. This function requires you to supply features for scoring. It will run ModelDataPrep()to prepare your features for H2O data conversion and scoring.

Usage

```
AutoH2OMLScoring(ScoringData = NULL, FeatureColumnNames = NULL,
  ModelType = "mojo", H2OShutdown = TRUE, MaxMem = "28G",
  JavaOptions = "-Xmx1g -XX:ReservedCodeCacheSize=256m",
  ModelPath = NULL, ModelID = NULL, ReturnFeatures = TRUE,
  MDP_Impute = TRUE, MDP_CharToFactor = TRUE, MDP_RemoveDates = TRUE,
  MDP_MissFactor = "0", MDP_MissNum = -1)
```

Arguments

ScoringData	This is your data.table of features for scoring. Can be a single row or batch.
FeatureColumnNames	Supply either column names or column numbers used in the AutoH2o__() function
ModelType	Set to either "mojo" or "standard" depending on which version you saved
H2OShutdown	Set to TRUE is you are scoring a "standard" model file and you aren't planning on continuing to score.
MaxMem	Set to you dedicated amount of memory. E.g. "28G"
JavaOptions	Change the default to your machines specification if needed. Default is '-Xmx1g -XX:ReservedCodeCacheSize=256m',
ModelPath	Supply your path file used in the AutoH2o__() function
ModelID	Supply the model ID used in the AutoH2o__() function
ReturnFeatures	Set to TRUE to return your features with the predicted values.
MDP_Impute	Set to TRUE if you did so for modeling and didn't do so before supplying ScoringData in this function
MDP_CharToFactor	Set to TRUE to turn your character columns to factors if you didn't do so to your ScoringData that you are supplying to this function
MDP_RemoveDates	Set to TRUE if you have date of timestamp columns in your ScoringData
MDP_MissFactor	If you set MDP_Impute to TRUE, supply the character values to replace missing values with
MDP_MissNum	If you set MDP_Impute to TRUE, supply a numeric value to replace missing values with

Value

A data.table of predicted values with the option to return model features as well.

See Also

Other Supervised Learning: AutoCatBoostClassifier, AutoCatBoostMultiClass, AutoCatBoostRegression, AutoCatBoostScoring, AutoH2OModeler, AutoH2OScoring, AutoH2oDRFClassifier, AutoH2oDRFMultiClass, AutoH2oDRFRegression, AutoH2oGBMClassifier, AutoH2oGBMMultiClass, AutoH2oGBMRegression, AutoMLTS, AutoNLS, AutoRecommenderScoring, AutoRecommender, AutoTS, AutoXGBoostClassifier, AutoXGBoostMultiClass, AutoXGBoostRegression, AutoXGBoostScoring

Examples

```
Preds <- AutoH2OMLScoring(ScoringData = data,
                          FeatureColumnNames = 2:12,
                          ModelType = "mojo",
                          H2OShutdown = TRUE,
                          MaxMem = "28G",
                          JavaOptions = '-Xmx1g -XX:ReservedCodeCacheSize=256m',
                          ModelPath = NULL,
                          ModelID = "ModelTest",
                          ReturnFeatures = TRUE,
                          MDP_Impute = TRUE,
                          MDP_CharToFactor = TRUE,
                          MDP_RemoveDates = TRUE,
                          MDP_MissFactor = "0",
                          MDP_MissNum = -1)
```

AutoH2OModeler *An Automated Machine Learning Framework using H2O*

Description

Steps in the function include: See details below for information on using this function.

Usage

```
AutoH2OModeler(Construct, max_memory = "28G", ratios = 0.8,
  BL_Trees = 500, nthreads = 1, model_path = NULL,
  MaxRuntimeSeconds = 3600, MaxModels = 30, TrainData = NULL,
  TestData = NULL, SaveToFile = FALSE, ReturnObjects = TRUE)
```

Arguments

Construct	Core instruction file for automation (see Details below for more information on this)
max_memory	The ceiling amount of memory H2O will utilize
ratios	The percentage of train samples from source data (remainder goes to validation set)
BL_Trees	The number of trees to build in baseline GBM or RandomForest

`nthreads` Set the number of threads to run function

`model_path` Directory path for where you want your models saved

`MaxRuntimeSeconds` Number of seconds of run time for grid tuning

`MaxModels` Number of models you'd like to have returned

`TrainData` Set to NULL or supply a data.table for training data

`TestData` Set to NULL or supply a data.table for validation data

`SaveToFile` Set to TRUE to save models and output to model_path

`ReturnObjects` Set to TRUE to return objects from functioin

Details

1. Logic: Error checking in the modeling arguments from your Construction file

2. ML: Build grid-tuned models and baseline models for comparison and checks which one performs better on validation data

3. Evaluation: Collects the performance metrics for both

4. Evaluation: Generates calibration plots (and boxplots for regression) for the winning model

5. Evaluation: Generates partial dependence calibration plots (and boxplots for regression) for the winning model

6. Evaluation: Generates variable importance tables and a table of non-important features

7. Production: Creates a storage file containing: model name, model path, grid tune performance, baseline performance, and threshold (if classification) and stores that file in your model_path location

The Construct file must be a data.table and the columns need to be in the correct order (see examples). Character columns must be converted to type "Factor". You must remove date columns or convert them to "Factor". For classification models, your target variable needs to be a (0,1) of type "Factor." See the examples below for help with setting up the Construct file for various modeling target variable types. There are examples for regression, classification, multinomial, and quantile regression. For help on which parameters to use, look up the r/h2o documentation. If you misspecify the construct file, it will produce an error and outputfile of what was wrong and suggestions for fixing the error.

Let's go over the construct file, column by column. The Targets column is where you specify the column number of your target variable (in quotes, e.g. "c(1)").

The Distribution column is where you specify the distribution type for the modeling task. For classification use bernoulli, for multilabel use multinomial, for quantile use quantile, and for regression, you can choose from the list available in the H2O docs, such as gaussian, poisson, gamma, etc. It's not set up to handle tweedie distributions currently but I can add support if there is demand.

The Loss column tells H2O which metric to use for the loss metrics. For regression, I typically use "mse", quantile regression, "mae", classification "auc", and multinomial "logloss". For deeplearning models, you need to use "quadratic", "absolute", and "crossentropy".

The Quantile column tells H2O which quantile to use for quantile regression (in decimal form).

The ModelName column is the name you wish to give your model as a prefix.

The Algorithm column is the model you wish to use: gbm, randomForest, deeplearning, AutoML, XGBoost, LightGBM.

The dataName column is the name of your data.

The TargetCol column is the column number of your target variable.

The FeatureCols column is the column numbers of your features.

The CreateDate column is for tracking your model build dates.

The GridTune column is a TRUE / FALSE column for whether you want to run a grid tune model for comparison.

The ExportValidData column is a TRUE / FALSE column indicating if you want to export the validation data.

The ParDep column is where you put the number of partial dependence calibration plots you wish to generate.

The PD_Data column is where you specify if you want to generate the partial dependence plots on "All" data, "Validate" data, or "Train" data.

The ThreshType column is for classification models. You can specify "f1", "f2", "f0point5", or "CS" for cost sentitive.

The FSC column is the feature selection column. Specify the percentage importance cutoff to create a table of "unimportant" features.

The tpProfit column is for when you specify "CS" in the ThreshType column. This is your true positive profit.

The tnProfit column is for when you specify "CS" in the ThreshType column. This is your true negative profit.

The fpProfit column is for when you specify "CS" in the ThreshType column. This is your false positive profit.

The fnProfit column is for when you specify "CS" in the ThreshType column. This is your false negative profit.

The SaveModel column is a TRUE / FALSE indicator. If you are just testing out models, set this to FALSE.

The SaveModelType column is where you specify if you want a "standard" model object saveed or a "mojo" model object saved.

The PredsAllData column is a TRUE / FALSE column. Set to TRUE if you want all the predicted values returns (for all data).

The TargetEncoding column let's you specify the column number of features you wish to run target encoding on. Set to NA to not run this feature.

The SupplyData column lets you supply the data names for training and validation data. Set to NULL if you want the data partitioning to be done internally.

Value

Returns saved models, corrected Construct file, variable importance tables, evaluation and partial dependence calibration plots, model performance measure, and a file called grid_tuned_paths.Rdata which contains paths to your saved models for operationalization.

Author(s)

Adrian Antico

See Also

Other Supervised Learning: AutoCatBoostClassifier, AutoCatBoostMultiClass, AutoCatBoostRegression, AutoCatBoostScoring, AutoH2OMLScoring, AutoH2OScoring, AutoH2oDRFClassifier, AutoH2oDRFMultiClass, AutoH2oDRFRegression, AutoH2oGBMClassifier, AutoH2oGBMMultiClass, AutoH2oGBMRegression, AutoMLTS, AutoNLS, AutoRecommenderScoring, AutoRecommender, AutoTS, AutoXGBoostClassifier, AutoXGBoostMultiClass, AutoXGBoostRegression, AutoXGBoostScoring

Examples

```
# Classification Example
Correl <- 0.85
aa <- data.table::data.table(target = runif(1000))
aa[, x1 := qnorm(target)]
aa[, x2 := runif(1000)]
aa[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                          sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable2 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable3 := exp(pnorm(Correl * x1 +
                                          sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                              sqrt(1-Correl^2) * qnorm(x2))))]
aa[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                           sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable6 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^0.10]
aa[, Independent_Variable7 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^0.25]
aa[, Independent_Variable8 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^0.75]
aa[, Independent_Variable9 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^2]
aa[, Independent_Variable10 := (pnorm(Correl * x1 +
                                        sqrt(1-Correl^2) * qnorm(x2)))^4]
aa[, ':=' (x1 = NULL, x2 = NULL)]
aa[, target := as.factor(ifelse(target > 0.5,1,0))]
Construct <- data.table::data.table(Targets = rep("target",3),
                                    Distribution    = c("bernoulli",
                                                        "bernoulli",
                                                        "bernoulli"),
                                    Loss            = c("AUC","AUC","CrossEntropy"),
                                    Quantile        = rep(NA,3),
                                    ModelName       = c("GBM","DRF","DL"),
                                    Algorithm       = c("gbm",
                                                        "randomForest",
                                                        "deeplearning"),
                                    dataName        = rep("aa",3),
                                    TargetCol       = rep(c("1"),3),
                                    FeatureCols     = rep(c("2:11"),3),
                                    CreateDate      = rep(Sys.time(),3),
                                    GridTune        = rep(FALSE,3),
                                    ExportValidData = rep(TRUE,3),
                                    ParDep          = rep(2,3),
                                    PD_Data         = rep("All",3),
                                    ThreshType      = rep("f1",3),
```

```
                                    FSC             = rep(0.001,3),
                                    tpProfit        = rep(NA,3),
                                    tnProfit        = rep(NA,3),
                                    fpProfit        = rep(NA,3),
                                    fnProfit        = rep(NA,3),
                                    SaveModel       = rep(FALSE,3),
                                    SaveModelType   = c("Mojo","standard","mojo"),
                                    PredsAllData    = rep(TRUE,3),
                                    TargetEncoding  = rep(NA,3),
                                    SupplyData      = rep(FALSE,3))
AutoH2OModeler(Construct,
               max_memory = "28G",
               ratios = 0.75,
               BL_Trees = 500,
               nthreads = 5,
               model_path = NULL,
               MaxRuntimeSeconds = 3600,
               MaxModels = 30,
               TrainData = NULL,
               TestData  = NULL,
               SaveToFile = FALSE,
               ReturnObjects = TRUE)

# Multinomial Example
Correl <- 0.85
aa <- data.table::data.table(target = runif(1000))
aa[, x1 := qnorm(target)]
aa[, x2 := runif(1000)]
aa[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                          sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable2 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable3 := exp(pnorm(Correl * x1 +
                                          sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                              sqrt(1-Correl^2) * qnorm(x2))))]
aa[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                           sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable6 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^0.10]
aa[, Independent_Variable7 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^0.25]
aa[, Independent_Variable8 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^0.75]
aa[, Independent_Variable9 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^2]
aa[, Independent_Variable10 := (pnorm(Correl * x1 +
                                        sqrt(1-Correl^2) * qnorm(x2)))^4]
aa[, ':=' (x1 = NULL, x2 = NULL)]
aa[, target := as.factor(ifelse(target < 0.33,"A",ifelse(target < 0.66, "B","C")))]
Construct <- data.table::data.table(Targets = rep("target",3),
                                    Distribution    = c("multinomial",
                                                        "multinomial",
                                                        "multinomial"),
                                    Loss            = c("auc","logloss","accuracy"),
                                    Quantile        = rep(NA,3),
                                    ModelName       = c("GBM","DRF","DL"),
```

```
                                    Algorithm       = c("gbm",
                                                        "randomForest",
                                                        "deeplearning"),
                                    dataName        = rep("aa",3),
                                    TargetCol       = rep(c("1"),3),
                                    FeatureCols     = rep(c("2:11"),3),
                                    CreateDate      = rep(Sys.time(),3),
                                    GridTune        = rep(FALSE,3),
                                    ExportValidData = rep(TRUE,3),
                                    ParDep          = rep(NA,3),
                                    PD_Data         = rep("All",3),
                                    ThreshType      = rep("f1",3),
                                    FSC             = rep(0.001,3),
                                    tpProfit        = rep(NA,3),
                                    tnProfit        = rep(NA,3),
                                    fpProfit        = rep(NA,3),
                                    fnProfit        = rep(NA,3),
                                    SaveModel       = rep(FALSE,3),
                                    SaveModelType   = c("Mojo","standard","mojo"),
                                    PredsAllData    = rep(TRUE,3),
                                    TargetEncoding  = rep(NA,3),
                                    SupplyData      = rep(FALSE,3))

AutoH2OModeler(Construct,
               max_memory = "28G",
               ratios = 0.75,
               BL_Trees = 500,
               nthreads = 5,
               model_path = NULL,
               MaxRuntimeSeconds = 3600,
               MaxModels = 30,
               TrainData = NULL,
               TestData  = NULL,
               SaveToFile = FALSE,
               ReturnObjects = TRUE)

# Regression Example
Correl <- 0.85
aa <- data.table::data.table(target = runif(1000))
aa[, x1 := qnorm(target)]
aa[, x2 := runif(1000)]
aa[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                          sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable2 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable3 := exp(pnorm(Correl * x1 +
                                          sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                              sqrt(1-Correl^2) * qnorm(x2))))]
aa[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                           sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable6 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^0.10]
aa[, Independent_Variable7 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^0.25]
aa[, Independent_Variable8 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^0.75]
```

```
aa[, Independent_Variable9 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^2]
aa[, Independent_Variable10 := (pnorm(Correl * x1 +
                                        sqrt(1-Correl^2) * qnorm(x2)))^4]
aa[, ':=' (x1 = NULL, x2 = NULL)]
Construct <- data.table::data.table(Targets = rep("target",3),
                                    Distribution    = c("gaussian",
                                                        "gaussian",
                                                        "gaussian"),
                                    Loss            = c("MSE","MSE","Quadratic"),
                                    Quantile        = rep(NA,3),
                                    ModelName       = c("GBM","DRF","DL"),
                                    Algorithm       = c("gbm",
                                                        "randomForest",
                                                        "deeplearning"),
                                    dataName        = rep("aa",3),
                                    TargetCol       = rep(c("1"),3),
                                    FeatureCols     = rep(c("2:11"),3),
                                    CreateDate      = rep(Sys.time(),3),
                                    GridTune        = rep(FALSE,3),
                                    ExportValidData = rep(TRUE,3),
                                    ParDep          = rep(2,3),
                                    PD_Data         = rep("All",3),
                                    ThreshType      = rep("f1",3),
                                    FSC             = rep(0.001,3),
                                    tpProfit        = rep(NA,3),
                                    tnProfit        = rep(NA,3),
                                    fpProfit        = rep(NA,3),
                                    fnProfit        = rep(NA,3),
                                    SaveModel       = rep(FALSE,3),
                                    SaveModelType   = c("Mojo","standard","mojo"),
                                    PredsAllData    = rep(TRUE,3),
                                    TargetEncoding  = rep(NA,3),
                                    SupplyData      = rep(FALSE,3))
AutoH2OModeler(Construct,
               max_memory = "28G",
               ratios = 0.75,
               BL_Trees = 500,
               nthreads = 5,
               model_path = NULL,
               MaxRuntimeSeconds = 3600,
               MaxModels = 30,
               TrainData = NULL,
               TestData  = NULL,
               SaveToFile = FALSE,
               ReturnObjects = TRUE)

# Quantile Regression Example
Correl <- 0.85
aa <- data.table::data.table(target = runif(1000))
aa[, x1 := qnorm(target)]
aa[, x2 := runif(1000)]
aa[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                          sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable2 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable3 := exp(pnorm(Correl * x1 +
```

```
                                          sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                              sqrt(1-Correl^2) * qnorm(x2))))]
aa[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                           sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable6 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^0.10]
aa[, Independent_Variable7 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^0.25]
aa[, Independent_Variable8 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^0.75]
aa[, Independent_Variable9 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^2]
aa[, Independent_Variable10 := (pnorm(Correl * x1 +
                                        sqrt(1-Correl^2) * qnorm(x2)))^4]
aa[, ':=' (x1 = NULL, x2 = NULL)]
Construct <- data.table::data.table(Targets = rep("target",3),
                                    Distribution    = c("quantile",
                                                        "quantile"),
                                    Loss            = c("MAE","Absolute"),
                                    Quantile        = rep(0.75,2),
                                    ModelName       = c("GBM","DL"),
                                    Algorithm       = c("gbm",
                                                        "deeplearning"),
                                    dataName        = rep("aa",2),
                                    TargetCol       = rep(c("1"),2),
                                    FeatureCols     = rep(c("2:11"),2),
                                    CreateDate      = rep(Sys.time(),2),
                                    GridTune        = rep(FALSE,2),
                                    ExportValidData = rep(TRUE,2),
                                    ParDep          = rep(4,2),
                                    PD_Data         = rep("All",2),
                                    ThreshType      = rep("f1",2),
                                    FSC             = rep(0.001,2),
                                    tpProfit        = rep(NA,2),
                                    tnProfit        = rep(NA,2),
                                    fpProfit        = rep(NA,2),
                                    fnProfit        = rep(NA,2),
                                    SaveModel       = rep(FALSE,2),
                                    SaveModelType   = c("Mojo","mojo"),
                                    PredsAllData    = rep(TRUE,2),
                                    TargetEncoding  = rep(NA,2),
                                    SupplyData      = rep(FALSE,2))
AutoH2OModeler(Construct,
               max_memory = "28G",
               ratios = 0.75,
               BL_Trees = 500,
               nthreads = 5,
               model_path = NULL,
               MaxRuntimeSeconds = 3600,
               MaxModels = 30,
               TrainData = NULL,
               TestData  = NULL,
               SaveToFile = FALSE,
               ReturnObjects = TRUE)
```

AutoH2OScoring *AutoH2OScoring is the complement of AutoH20Modeler.*

Description

AutoH2OScoring is the complement of AutoH20Modeler. Use this for scoring models. You can score regression, quantile regression, classification, multinomial, clustering, and text models (built with the Word2VecModel function). You can also use this to score multioutcome models so long as the there are two models: one for predicting the count of outcomes (a count outcome in character form) and a multinomial model on the label data. You will want to ensure you have a record for each label in your training data in (0,1) as factor form.

Usage

```
AutoH2OScoring(Features = data, GridTuneRow = c(1:3),
  ScoreMethod = "Standard", TargetType = rep("multinomial", 3),
  ClassVals = rep("probs", 3), NThreads = 6, MaxMem = "28G",
  JavaOptions = "-Xmx1g -XX:ReservedCodeCacheSize=256m",
  SaveToFile = FALSE, FilesPath = NULL, H20ShutDown = rep(FALSE, 3))
```

Arguments

`Features`	This is a data.table of features for scoring.
`GridTuneRow`	Numeric. The row numbers of grid_tuned_paths, KMeansModelFile, or StoreFile containing the model you wish to score
`ScoreMethod`	"Standard" or "Mojo": Mojo is available for supervised models; use standard for all others
`TargetType`	"Regression", "Classification", "Multinomial", "MultiOutcome", "Text", "Clustering". MultiOutcome must be two multinomial models, a count model (the count of outcomes, as a character value), and the multinomial model predicting the labels.
`ClassVals`	Choose from "p1", "Probs", "Label", or "All" for classification and multinomial models.
`NThreads`	Number of available threads for H2O
`MaxMem`	Amount of memory to dedicate to H2O
`JavaOptions`	Modify to your machine if the default doesn't work
`SaveToFile`	Set to TRUE if you want your model scores saved to file.
`FilesPath`	Set this to the folder where your models and model files are saved
`H20ShutDown`	TRUE to shutdown H2O after the run. Use FALSE if you will be repeatedly scoring and shutdown somewhere else in your environment.

Value

Returns a list of predicted values. Each list element contains the predicted values from a single model predict call.

Author(s)

Adrian Antico

See Also

Other Supervised Learning: AutoCatBoostClassifier, AutoCatBoostMultiClass, AutoCatBoostRegression, AutoCatBoostScoring, AutoH2OMLScoring, AutoH2OModeler, AutoH2oDRFClassifier, AutoH2oDRFMultiClass, AutoH2oDRFRegression, AutoH2oGBMClassifier, AutoH2oGBMMultiClass, AutoH2oGBMRegression, AutoMLTS, AutoNLS, AutoRecommenderScoring, AutoRecommender, AutoTS, AutoXGBoostClassifier, AutoXGBoostMultiClass, AutoXGBoostRegression, AutoXGBoostScoring

Examples

```
# Multinomial Example
Correl <- 0.85
aa <- data.table::data.table(target = runif(1000))
aa[, x1 := qnorm(target)]
aa[, x2 := runif(1000)]
aa[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                          sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable2 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable3 := exp(pnorm(Correl * x1 +
                                          sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                              sqrt(1-Correl^2) * qnorm(x2))))]
aa[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                           sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable6 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^0.10]
aa[, Independent_Variable7 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^0.25]
aa[, Independent_Variable8 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^0.75]
aa[, Independent_Variable9 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^2]
aa[, Independent_Variable10 := (pnorm(Correl * x1 +
                                        sqrt(1-Correl^2) * qnorm(x2)))^4]
aa[, ':=' (x1 = NULL, x2 = NULL)]
aa[, target := as.factor(ifelse(target < 0.33,"A",ifelse(target < 0.66, "B","C")))]
Construct <- data.table::data.table(Targets = rep("target",3),
                                    Distribution    = c("multinomial",
                                                        "multinomial",
                                                        "multinomial"),
                                 Loss           = c("logloss","logloss","CrossEntropy"),
                                    Quantile        = rep(NA,3),
                                    ModelName       = c("GBM","DRF","DL"),
                                    Algorithm       = c("gbm",
                                                        "randomForest",
                                                        "deeplearning"),
                                    dataName        = rep("aa",3),
                                    TargetCol       = rep(c("1"),3),
                                    FeatureCols     = rep(c("2:11"),3),
                                    CreateDate      = rep(Sys.time(),3),
                                    GridTune        = rep(FALSE,3),
```

```
                                        ExportValidData = rep(TRUE,3),
                                        ParDep          = rep(NA,3),
                                        PD_Data         = rep("All",3),
                                        ThreshType      = rep("f1",3),
                                        FSC             = rep(0.001,3),
                                        tpProfit        = rep(NA,3),
                                        tnProfit        = rep(NA,3),
                                        fpProfit        = rep(NA,3),
                                        fnProfit        = rep(NA,3),
                                        SaveModel       = rep(FALSE,3),
                                        SaveModelType   = c("Mojo","mojo","mojo"),
                                        PredsAllData    = rep(TRUE,3),
                                        TargetEncoding  = rep(NA,3),
                                        SupplyData      = rep(FALSE,3))

AutoH2OModeler(Construct,
               max_memory = "28G",
               ratios = 0.75,
               BL_Trees = 500,
               nthreads = 5,
               model_path = NULL,
               MaxRuntimeSeconds = 3600,
               MaxModels = 30,
               TrainData = NULL,
               TestData  = NULL,
               SaveToFile = FALSE,
               ReturnObjects = TRUE)

N <- 3
data <- AutoH2OScoring(Features     = aa,
                       GridTuneRow  = c(1:N),
                       ScoreMethod  = "standard",
                       TargetType   = rep("multinomial",N),
                       ClassVals    = rep("Probs",N),
                       NThreads     = 6,
                       MaxMem       = "28G",
                       JavaOptions  = '-Xmx1g -XX:ReservedCodeCacheSize=256m',
                       SaveToFile   = FALSE,
                       FilesPath    = NULL,
                       H20ShutDown  = rep(FALSE,N))
```

AutoH2OTextPrepScoring *AutoH2OTextPrepScoring is for NLP scoring*

Description

This function returns prepared tokenized data for H2O Word2VecModeler scoring

Usage

```
AutoH2OTextPrepScoring(data, string, MaxMem, NThreads)
```

Arguments

`data`	The text data
`string`	The name of the string column to prepare
`MaxMem`	Amount of memory you want to let H2O utilize
`NThreads`	The number of threads you want to let H2O utilize

Author(s)

Adrian Antico

See Also

Other Misc: AutoRecomDataCreate, ChartTheme, PrintObjectsSize, RemixTheme, SimpleCap, multiplot, percRank, tempDatesFun, tokenizeH2O

Examples

```
data <- AutoH2OTextPrepScoring(data = x,
                               string = "text_column",
                               MaxMem = "28G",
                               NThreads = 8)
```

AutoKMeans — *AutoKMeans Automated row clustering for mixed column types*

Description

AutoKMeans adds a column to your original data with a cluster number identifier. Uses glrm (grid tune-able) and then k-means to find optimal k.

Usage

```
AutoKMeans(data, nthreads = 8, MaxMem = "28G", SaveModels = NULL,
  PathFile = NULL, GridTuneGLRM = TRUE, GridTuneKMeans = TRUE,
  glrmCols = c(1:5), IgnoreConstCols = TRUE, glrmFactors = 5,
  Loss = "Absolute", glrmMaxIters = 1000, SVDMethod = "Randomized",
  MaxRunTimeSecs = 3600, KMeansK = 50, KMeansMetric = "totss")
```

Arguments

`data`	is the source time series data.table
`nthreads`	set based on number of threads your machine has available
`MaxMem`	set based on the amount of memory your machine has available
`SaveModels`	Set to "standard", "mojo", or NULL (default)
`PathFile`	Set to folder where you will keep the models
`GridTuneGLRM`	If you want to grid tune the glrm model, set to TRUE, FALSE otherwise
`GridTuneKMeans`	If you want to grid tuen the KMeans model, set to TRUE, FALSE otherwise

`glrmCols` the column numbers for the glrm

`IgnoreConstCols` tell H2O to ignore any columns that have zero variance

`glrmFactors` similar to the number of factors to return from PCA

`Loss` set to one of "Quadratic", "Absolute", "Huber", "Poisson", "Hinge", "Logistic", "Periodic"

`glrmMaxIters` max number of iterations

`SVDMethod` choose from "Randomized","GramSVD","Power"

`MaxRunTimeSecs` set the timeout for max run time

`KMeansK` number of factors to test out in k-means to find the optimal number

`KMeansMetric` pick the metric to identify top model in grid tune c("totss","betweenss","withinss")

Value

Original data.table with added column with cluster number identifier

Author(s)

Adrian Antico

See Also

Other Unsupervised Learning: GenTSAnomVars, ResidualOutliers

Examples

```
data <- data.table::as.data.table(iris)
data <- AutoKMeans(data,
                   nthreads = 8,
                   MaxMem = "28G",
                   SaveModels = NULL,
                   PathFile = NULL,
                   GridTuneGLRM = TRUE,
                   GridTuneKMeans = TRUE,
                   glrmCols = 1:(ncol(data)-1),
                   IgnoreConstCols = TRUE,
                   glrmFactors = 2,
                   Loss = "Absolute",
                   glrmMaxIters = 1000,
                   SVDMethod = "Randomized",
                   MaxRunTimeSecs = 3600,
                   KMeansK = 5,
                   KMeansMetric = "totss")
unique(data[["Species"]])
unique(data[["ClusterID"]])
temp <- data[, mean(ClusterID), by = "Species"]
Setosa <- round(temp[Species == "setosa", V1][[1]],0)
Versicolor <- round(temp[Species == "versicolor", V1][[1]],0)
Virginica <- round(temp[Species == "virginica", V1][[1]],0)
data[, Check := "a"]
data[ClusterID == eval(Setosa), Check := "setosa"]
data[ClusterID == eval(Virginica), Check := "virginica"]
data[ClusterID == eval(Versicolor), Check := "versicolor"]
```

```
data[, Acc := as.numeric(ifelse(Check == Species, 1, 0))]
data[, mean(Acc)][[1]]
```

AutoMLTS *AutoMLTS Is an Automated Machine Learning Time Series Forecasting Function*

Description

AutoMLTS Is an Automated Machine Learning Time Series Forecasting Function. Create hundreds of thousands of time series forecasts using this function.

Usage

```
AutoMLTS(data, TargetColumnName = "Target",
  DateColumnName = "DateTime", GroupVariables = NULL,
  FC_Periods = 30, TimeUnit = "week", Lags = c(1:5),
  MA_Periods = c(1:5), CalendarVariables = FALSE,
  TimeTrendVariable = FALSE, DataTruncate = FALSE,
  SplitRatios = c(0.7, 0.2, 0.1), TaskType = "GPU",
  EvalMetric = "MAPE", GridTune = FALSE, GridEvalMetric = "mape",
  ModelCount = 1, ModelType = "catboost", NTrees = 1000,
  PartitionType = "timeseries", Timer = TRUE)
```

Arguments

`data`	Supply your full series data set here
`TargetColumnName`	List the column name of your target variables column. E.g. "Target"
`DateColumnName`	List the column name of your date column. E.g. "DateTime"
`GroupVariables`	Defaults to NULL. Use NULL when you have a single series. Add in GroupVariables when you have a series for every level of a group or multiple groups.
`FC_Periods`	Set the number of periods you want to have forecasts for. E.g. 52 for weekly data to forecast a year ahead
`TimeUnit`	List the time unit your data is aggregated by. E.g. "hour", "day", "week", "year"
`Lags`	Select the periods for all lag variables you want to create. E.g. I use this for weekly data c(1:5,52)
`MA_Periods`	Select the periods for all moving average variables you want to create. E.g. I use this for weekly data c(1:5,52)
`CalendarVariables`	Set to TRUE to have calendar variables created. The calendar variables are numeric representations of second, minute, hour, week day, month day, year day, week, isoweek, quarter, and year
`TimeTrendVariable`	Set to TRUE to have a time trend variable added to the model. Time trend is numeric variable indicating the numeric value of each record in the time series (by group). Time trend starts at 1 for the earliest point in time and increments by one for each success time point.

DataTruncate	Set to TRUE to remove records with missing values from the lags and moving average features created
SplitRatios	E.g c(0.7,0.2,0.1) for train, validation, and test sets
TaskType	Default is "GPU" but you can also set it to "CPU"
EvalMetric	Select from "RMSE", "MAE", "MAPE", "Poisson", "Quantile", "LogLinQuantile", "Lq", "NumErrors", "SMAPE", "R2", "MSLE", "MedianAbsoluteError"
GridTune	Set to TRUE to run a grid tune
GridEvalMetric	This is the metric used to find the threshold 'poisson', 'mae', 'mape', 'mse', 'msle', 'kl', 'cs', 'r2'
ModelCount	Set the number of models to try in the grid tune
ModelType	Select from list "catboost"
NTrees	Select the number of trees you want to have built to train the model
PartitionType	Select "random" for random data partitioning "time" for partitioning by time frames
Timer	= TRUE

Value

Returns a data.table of original series and forecasts, the catboost model objects (everything returned from AutoCatBoostRegression()), and a time series forecast plot. The time series forecast plot will plot your single series or aggregate your data to a single series and create a plot from that.

See Also

Other Supervised Learning: AutoCatBoostClassifier, AutoCatBoostMultiClass, AutoCatBoostRegression, AutoCatBoostScoring, AutoH2OMLScoring, AutoH2OModeler, AutoH2OScoring, AutoH2oDRFClassifier, AutoH2oDRFMultiClass, AutoH2oDRFRegression, AutoH2oGBMClassifier, AutoH2oGBMMultiClass, AutoH2oGBMRegression, AutoNLS, AutoRecommenderScoring, AutoRecommender, AutoTS, AutoXGBoostClassifier, AutoXGBoostMultiClass, AutoXGBoostRegression, AutoXGBoostScoring

Examples

```
Results <- AutoMLTS(data,
                    TargetColumnName = "Weekly_Sales",
                    DateColumnName = "Date",
                    GroupVariables = c("Store","Dept"),
                    FC_Periods = 52,
                    TimeUnit = "week",
                    Lags = c(1:5,52),
                    MA_Periods = c(1:5,52),
                    CalendarVariables = TRUE,
                    TimeTrendVariable = TRUE,
                    DataTruncate = FALSE,
                    SplitRatios = c(1-2*30/143,30/143,30/143),
                    TaskType = "GPU",
                    EvalMetric = "MAE",
                    GridTune = FALSE,
                    GridEvalMetric = "mae",
                    ModelCount = 1,
```

```
                        ModelType = "catboost",
                        NTrees = 1000,
                        PartitionType = "time")
Results$TimeSeriesPlot
Results$Forecast
Results$ModelInformation$...
```

AutoNLS *AutoNLS is a function for automatically building nls models*

Description

This function will build models for 9 different nls models, along with a non-parametric monotonic regression and a polynomial regression. The models are evaluated, a winner is picked, and the predicted values are stored in your data table.

Usage

```
AutoNLS(data, y, x, monotonic = TRUE)
```

Arguments

data	Data is the data table you are building the modeling on
y	Y is the target variable name in quotes
x	X is the independent variable name in quotes
monotonic	This is a TRUE/FALSE indicator - choose TRUE if you want monotonic regression over polynomial regression

Value

A list containing "PredictionData" which is a data table with your original column replaced by the nls model predictions; "ModelName" the model name; "ModelObject" The winning model to later use; "EvaluationMetrics" Model metrics for models with ability to build.

Author(s)

Adrian Antico

See Also

Other Supervised Learning: AutoCatBoostClassifier, AutoCatBoostMultiClass, AutoCatBoostRegression, AutoCatBoostScoring, AutoH2OMLScoring, AutoH2OModeler, AutoH2OScoring, AutoH2oDRFClassifier, AutoH2oDRFMultiClass, AutoH2oDRFRegression, AutoH2oGBMClassifier, AutoH2oGBMMultiClass, AutoH2oGBMRegression, AutoMLTS, AutoRecommenderScoring, AutoRecommender, AutoTS, AutoXGBoostClassifie
AutoXGBoostMultiClass, AutoXGBoostRegression, AutoXGBoostScoring

Examples

```
# Create Growth Data
data <-
  data.table::data.table(Target = seq(1, 500, 1),
                         Variable = rep(1, 500))
for (i in as.integer(1:500)) {
  if (i == 1) {
    var <- data[i, "Target"][[1]]
    data.table::set(data,
                    i = i,
                    j = 2L,
                    value = var * (1 + runif(1) / 100))
  } else {
    var <- data[i - 1, "Variable"][[1]]
    data.table::set(data,
                    i = i,
                    j = 2L,
                    value = var * (1 + runif(1) / 100))
  }
}

# Add jitter to Target
data[, Target := jitter(Target,
                        factor = 0.25)]

# To keep original values
data1 <- data.table::copy(data)

# Merge and Model data
data11 <- AutoNLS(
  data = data,
  y = "Target",
  x = "Variable",
  monotonic = TRUE
)

# Join predictions to source data
data2 <- merge(
  data1,
  data11$PredictionData,
  by = "Variable",
  all = FALSE
)

# Plot output
ggplot2::ggplot(data2, ggplot2::aes(x = Variable)) +
  ggplot2::geom_line(ggplot2::aes(y = data2[["Target.x"]],
                                  color = "Target")) +
  ggplot2::geom_line(ggplot2::aes(y = data2[["Target.y"]],
                                  color = "Predicted")) +
 RemixAutoML::ChartTheme(Size = 12) +
  ggplot2::ggtitle(paste0("Growth Models AutoNLS: ",
                          data11$ModelName)) +
  ggplot2::ylab("Target Variable") +
  ggplot2::xlab("Independent Variable") +
  ggplot2::scale_colour_manual("Values",
```

```
                                    breaks = c("Target",
                                               "Predicted"),
                                    values = c("red",
                                               "blue"))
summary(data11$ModelObject)
data11$EvaluationMetrics
```

AutoRecomDataCreate *Convert transactional data.table to a binary ratings matrix*

Description

Convert transactional data.table to a binary ratings matrix

Usage

```
AutoRecomDataCreate(data, EntityColName = "CustomerID",
  ProductColName = "StockCode", MetricColName = "TotalSales",
  ReturnMatrix = FALSE)
```

Arguments

data	This is your transactional data.table. Must include an Entity (typically customer), ProductCode (such as SKU), and a sales metric (such as total sales).
EntityColName	This is the column name in quotes that represents the column name for the Entity, such as customer
ProductColName	This is the column name in quotes that represents the column name for the product, such as SKU
MetricColName	This is the column name in quotes that represents the column name for the metric, such as total sales
ReturnMatrix	Set to FALSE to coerce the object (desired route) or TRUE to return a matrix

Value

A BinaryRatingsMatrix

Author(s)

Adrian Antico and Douglas Pestana

See Also

Other Misc: AutoH2OTextPrepScoring, ChartTheme, PrintObjectsSize, RemixTheme, SimpleCap, multiplot, percRank, tempDatesFun, tokenizeH2O

Examples

```
data <- data.table::data.table(CustomerID = c(1,1,2,2,3,3),
                               StockCode = c("A","B","A","A","B","A"),
                               TotalSales = c(2,3,4,5,1,2))
RatingsMatrix <- AutoRecomDataCreate(data,
                                     EntityColName = "CustomerID",
                                     ProductColName = "StockCode",
                                     MetricColName = "TotalSales",
                                     ReturnMatrix = TRUE)
```

`AutoRecommender` *Automatically build the best recommendere model among models available.*

Description

This function returns the winning model that you pass onto AutoRecommenderScoring

Usage

```
AutoRecommender(data, Partition = "Split", KFolds = 2, Ratio = 0.75,
  RatingType = "TopN", RatingsKeep = 20,
  SkipModels = "AssociationRules", ModelMetric = "TPR")
```

Arguments

`data`	This is your BinaryRatingsMatrix. See function RecomDataCreate
`Partition`	Choose from "split", "cross-validation", "bootstrap". See evaluationScheme in recommenderlab for details.
`KFolds`	Choose 2 for traditional train and test. Choose greater than 2 for the number of cross validations
`Ratio`	The ratio for train and test. E.g. 0.75 for 75 percent data allocated to training
`RatingType`	Choose from "TopN", "ratings", "ratingMatrix"
`RatingsKeep`	The total ratings you wish to return. Default is 20.
`SkipModels`	AssociationRules runs the slowest and may crash your system. Choose from: "AssociationRules","ItemBasedCF","UserBasedCF","PopularItems","RandomItems"
`ModelMetric`	Choose from "Precision", "Recall", "TPR", or "FPR"

Value

The winning model used for scoring in the AutoRecommenderScoring function

Author(s)

Adrian Antico and Douglas Pestana

See Also

Other Supervised Learning: AutoCatBoostClassifier, AutoCatBoostMultiClass, AutoCatBoostRegression, AutoCatBoostScoring, AutoH2OMLScoring, AutoH2OModeler, AutoH2OScoring, AutoH2oDRFClassifier, AutoH2oDRFMultiClass, AutoH2oDRFRegression, AutoH2oGBMClassifier, AutoH2oGBMMultiClass, AutoH2oGBMRegression, AutoMLTS, AutoNLS, AutoRecommenderScoring, AutoTS, AutoXGBoostClassifier, AutoXGBoostMultiClass, AutoXGBoostRegression, AutoXGBoostScoring

Examples

```
WinningModel <- AutoRecommender(RatingsMatrix,
                                Partition = "Split",
                                KFolds = 2,
                                Ratio = 0.75,
                                RatingType = "TopN",
                                RatingsKeep = 20,
                                SkipModels = "AssociationRules",
                                ModelMetric = "TPR")
```

AutoRecommenderScoring *The AutoRecomScoring function scores recommender models from AutoRecommender()*

Description

This function will take your ratings matrix and model and score your data in parallel.

Usage

```
AutoRecommenderScoring(data, WinningModel, EntityColName = "CustomerID",
  ProductColName = "StockCode")
```

Arguments

data	The binary ratings matrix from RecomDataCreate()
WinningModel	The winning model returned from AutoRecommender()
EntityColName	Typically your customer ID
ProductColName	Something like "StockCode"

Value

Returns the prediction data

Author(s)

Adrian Antico and Douglas Pestana

See Also

Other Supervised Learning: AutoCatBoostClassifier, AutoCatBoostMultiClass, AutoCatBoostRegression, AutoCatBoostScoring, AutoH2OMLScoring, AutoH2OModeler, AutoH2OScoring, AutoH2oDRFClassifier, AutoH2oDRFMultiClass, AutoH2oDRFRegression, AutoH2oGBMClassifier, AutoH2oGBMMultiClass, AutoH2oGBMRegression, AutoMLTS, AutoNLS, AutoRecommender, AutoTS, AutoXGBoostClassifier, AutoXGBoostMultiClass, AutoXGBoostRegression, AutoXGBoostScoring

Examples

```
# F(G(Z(x))): AutoRecommenderScoring(AutoRecommender(RecomDataCreate(TransactionData)))
Results <- AutoRecommenderScoring(
  data = AutoRecomDataCreate(
      data,
      EntityColName = "CustomerID",
      ProductColName = "StockCode",
      MetricColName = "TotalSales"),
  WinningModel = AutoRecommender(
      AutoRecomDataCreate(
        data,
        EntityColName = "CustomerID",
        ProductColName = "StockCode",
        MetricColName = "TotalSales"),
      Partition = "Split",
      KFolds = 2,
      Ratio = 0.75,
      RatingType = "TopN",
      RatingsKeep = 20,
      SkipModels = "AssociationRules",
      ModelMetric = "TPR"),
  EntityColName = "CustomerID",
  ProductColName = "StockCode")
```

AutoTS *AutoTS is an automated time series modeling function*

Description

Step 1 is to build all the models and evaluate them on the number of HoldOutPeriods periods you specify. Step 2 is to pick the winner and rebuild the winning model on the full data set. Step 3 is to generate forecasts with the final model for FCPeriods that you specify. AutoTS builds the best time series models for each type, using optimized box-cox transformations and using a user-supplied frequency for the ts data conversion along with a model-based frequency for the ts data conversion, compares all types, selects the winner, and generates a forecast. Models include:

Usage

```
AutoTS(data, TargetName = "Target", DateName = "DateTime",
  FCPeriods = 30, HoldOutPeriods = 30, EvaluationMetric = "MAPE",
  TimeUnit = "day", Lags = 25, SLags = 2, NumCores = 4,
  SkipModels = NULL, StepWise = TRUE, TSClean = TRUE,
  ModelFreq = TRUE, PrintUpdates = FALSE)
```

Arguments

`data`	is the source time series data as a data.table - or a data structure that can be converted to a data.table
`TargetName`	is the name of the target variable in your data.table
`DateName`	is the name of the date column in your data.table
`FCPeriods`	is the number of periods into the future you wish to forecast
`HoldOutPeriods`	is the number of periods to use for validation testing
`EvaluationMetric`	Set this to either "MAPE", "MSE", or "MAE". Default is "MAPE"
`TimeUnit`	is the level of aggregation your dataset comes in
`Lags`	is the number of lags you wish to test in various models (same as moving averages)
`SLags`	is the number of seasonal lags you wish to test in various models (same as moving averages)
`NumCores`	is the number of cores available on your computer
`SkipModels`	Don't run specified models - e.g. exclude all models "DSHW" "ARFIMA" "ARIMA" "ETS" "NNET" "TBATS" "TSLM"
`StepWise`	Set to TRUE to have ARIMA and ARFIMA run a stepwise selection process. Otherwise, all models will be generated in parallel execution, but still run much slower.
`TSClean`	Set to TRUE to have missing values interpolated and outliers replaced with interpolated values: creates separate models for a larger comparison set
`ModelFreq`	Set to TRUE to run a separate version of all models where the time series frequency is chosen algorithmically
`PrintUpdates`	Set to TRUE for a print to console of function progress

Details

DSHW: Double Seasonal Holt Winters

ARFIMA: Auto Regressive Fractional Integrated Moving Average

ARIMIA: Stepwise Auto Regressive Integrated Moving Average with specified max lags, seasonal lags, moving averages, and seasonal moving averages

ETS: Additive and Multiplicitive Exponential Smoothing and Holt Winters

NNetar: Auto Regressive Neural Network models automatically compares models with 1 lag or 1 seasonal lag compared to models with up to N lags and N seasonal lags

TBATS: Exponential smoothing state space model with Box-Cox transformation, ARMA errors, Trend and Seasonal components

TSLM: Time Series Linear Model - builds a linear model with trend and season components extracted from the data

Value

Returns a list containing 1: A data.table object with a date column and the forecasted values; 2: The model evaluation results; 3: The champion model for later use if desired; 4: The name of the champion model; 5. A time series ggplot with historical values and forecasted values.

Author(s)

Adrian Antico and Douglas Pestana

See Also

Other Supervised Learning: AutoCatBoostClassifier, AutoCatBoostMultiClass, AutoCatBoostRegression, AutoCatBoostScoring, AutoH2OMLScoring, AutoH2OModeler, AutoH2OScoring, AutoH2oDRFClassifier, AutoH2oDRFMultiClass, AutoH2oDRFRegression, AutoH2oGBMClassifier, AutoH2oGBMMultiClass, AutoH2oGBMRegression, AutoMLTS, AutoNLS, AutoRecommenderScoring, AutoRecommender, AutoXGBoostClassifier, AutoXGBoostMultiClass, AutoXGBoostRegression, AutoXGBoostScoring

Examples

```
data <- data.table::data.table(DateTime = as.Date(Sys.time()),
  Target = stats::filter(rnorm(100,
                             mean = 50,
                             sd = 20),
                       filter=rep(1,10),
                       circular=TRUE))
data[, temp := seq(1:100)][, DateTime := DateTime - temp][, temp := NULL]
data <- data[order(DateTime)]
output <-   AutoTS(data,
                   TargetName       = "Target",
                   DateName         = "DateTime",
                   FCPeriods        = 1,
                   HoldOutPeriods   = 1,
                   EvaluationMetric = "MAPE",
                   TimeUnit         = "day",
                   Lags             = 1,
                   SLags            = 1,
                   NumCores         = 4,
                   SkipModels       = c("NNET","TBATS","ETS","TSLM","ARFIMA","DSHW"),
                   StepWise         = TRUE,
                   TSClean          = FALSE,
                   ModelFreq        = TRUE,
                   PrintUpdates     = FALSE)
ForecastData <- output$Forecast
ModelEval    <- output$EvaluationMetrics
WinningModel <- output$TimeSeriesModel
```

AutoWord2VecModeler *Automated word2vec data generation via H2O*

Description

This function allows you to automatically build a word2vec model and merge the data onto your supplied dataset

Usage

```
AutoWord2VecModeler(data, stringCol = c("Text_Col1", "Text_Col2"),
  KeepStringCol = FALSE, model_path = NULL, vects = 100,
  SaveStopWords = FALSE, MinWords = 1, WindowSize = 12,
```

```
  Epochs = 25, StopWords = NULL, SaveModel = "standard",
  Threads = max(1, parallel::detectCores() - 2), MaxMemory = "28G",
  SaveOutput = FALSE)
```

Arguments

data	Source data table to merge vects onto
stringCol	A string name for the column to convert via word2vec
KeepStringCol	Set to TRUE if you want to keep the original string column that you convert via word2vec
model_path	A string path to the location where you want the model and metadata stored
vects	The number of vectors to retain from the word2vec model
SaveStopWords	Set to TRUE to save the stop words used
MinWords	For H2O word2vec model
WindowSize	For H2O word2vec model
Epochs	For H2O word2vec model
StopWords	For H2O word2vec model
SaveModel	Set to "standard" to save normally; set to "mojo" to save as mojo. NOTE: while you can save a mojo, I haven't figured out how to score it in the AutoH20Scoring function.
Threads	Number of available threads you want to dedicate to model building
MaxMemory	Amount of memory you want to dedicate to model building
SaveOutput	Set to TRUE to save your models to file

Author(s)

Adrian Antico

See Also

Other Feature Engineering: AutoDataPartition, CreateCalendarVariables, DT_GDL_Feature_Engineering, DummifyDT, FAST_GDL_Feature_Engineering, GDL_Feature_Engineering, ModelDataPrep, Scoring_GDL_Feature_E

Examples

```
data <- AutoWord2VecModeler(data,
                            stringCol = c("Text_Col1",
                                          "Text_Col2"),
                            KeepStringCol = FALSE,
                            model_path = NULL,
                            vects = 100,
                            SaveStopWords = FALSE,
                            MinWords = 1,
                            WindowSize = 1,
                            Epochs = 25,
                            StopWords = NULL,
                            SaveModel = "standard",
                            Threads = max(1,parallel::detectCores()-2),
                            MaxMemory = "28G",
                            SaveOutput = TRUE)
```

AutoWordFreq — *Automated Word Frequency and Word Cloud Creation*

Description

This function builds a word frequency table and a word cloud. It prepares data, cleans text, and generates output.

Usage

```
AutoWordFreq(data, TextColName = "DESCR",
  GroupColName = "ClusterAllNoTarget", GroupLevel = 0,
  RemoveEnglishStopwords = TRUE, Stemming = TRUE,
  StopWords = c("bla", "bla2"))
```

Arguments

`data`	Source data table
`TextColName`	A string name for the column
`GroupColName`	Set to NULL to ignore, otherwise set to Cluster column name (or factor column name)
`GroupLevel`	Must be set if GroupColName is defined. Set to cluster ID (or factor level)
`RemoveEnglishStopwords`	Set to TRUE to remove English stop words, FALSE to ignore
`Stemming`	Set to TRUE to run stemming on your text data
`StopWords`	Add your own stopwords, in vector format

Author(s)

Adrian Antico

See Also

Other EDA: `ProblematicFeatures`, `ProblematicRecords`

Examples

```
data <- data.table::data.table(
DESCR = c("Gru, Gru, Gru, Gru, Gru, Gru, Gru, Gru, Gru, Gru, Gru, Gru, Gru,
          Urkle, Urkle, Urkle, Urkle, Urkle, Urkle, Urkle, Gru, Gru, Gru,
          bears, bears, bears, bears, bears, bears, smug, smug, smug, smug,
          smug, smug, smug, smug, smug, smug, smug, smug, smug, smug, smug,
          eats, eats, eats, eats, eats, eats, beats, beats, beats, beats,
          beats, beats, beats, beats, beats, beats, beats, science, science,
          Dwigt, Dwigt, Dwigt, Dwigt, Dwigt, Dwigt, Dwigt, Dwigt, Dwigt, Dwigt,
          Schrute, Schrute, Schrute, Schrute, Schrute, Schrute, Schrute,
          James, James, James, James, James, James, James, James, James, James,
          Halpert, Halpert, Halpert, Halpert, Halpert, Halpert, Halpert, Halpert"))
data <- AutoWordFreq(data,
                     TextColName = "DESCR",
```

```
GroupColName = NULL,
GroupLevel = NULL,
RemoveEnglishStopwords = FALSE,
Stemming = FALSE,
StopWords = c("Bla"))
```

AutoXGBoostClassifier *AutoXGBoostClassifier is an automated XGBoost modeling framework with grid-tuning and model evaluation*

Description

AutoXGBoostClassifier is an automated XGBoost modeling framework with grid-tuning and model evaluation that runs a variety of steps. First, a stratified sampling (by the target variable) is done to create train and validation sets. Then, the function will run a random grid tune over N number of models and find which model is the best (a default model is always included in that set). Once the model is identified and built, several other outputs are generated: validation data with predictions, evaluation plot, evaluation boxplot, evaluation metrics, variable importance, partial dependence calibration plots, partial dependence calibration box plots, and column names used in model fitting.

Usage

```
AutoXGBoostClassifier(data, ValidationData = NULL, TestData = NULL,
  TargetColumnName = NULL, FeatureColNames = NULL, IDcols = NULL,
  eval_metric = "auc", Trees = 50, GridTune = FALSE,
  grid_eval_metric = "auc", TreeMethod = "hist",
  MaxModelsInGrid = 10, NThreads = 8, model_path = NULL,
  ModelID = "FirstModel", NumOfParDepPlots = 3, Verbose = 0,
  ReturnModelObjects = TRUE, SaveModelObjects = FALSE,
  PassInGrid = NULL)
```

Arguments

`data`	This is your data set for training and testing your model
`ValidationData`	This is your holdout data set used in modeling either refine your hyperparameters.
`TestData`	This is your holdout data set. Catboost using both training and validation data in the training process so you should evaluate out of sample performance with this data set.
`TargetColumnName`	Either supply the target column name OR the column number where the target is located (but not mixed types). Note that the target column needs to be a 0 — 1 numeric variable.
`FeatureColNames`	Either supply the feature column names OR the column number where the target is located (but not mixed types)
`IDcols`	A vector of column names or column numbers to keep in your data but not include in the modeling.
`eval_metric`	This is the metric used to identify best grid tuned model. Choose from "logloss","error","aucpr","auc"

- `Trees` The maximum number of trees you want in your models
- `GridTune` Set to TRUE to run a grid tuning procedure. Set a number in MaxModelsInGrid to tell the procedure how many models you want to test.
- `grid_eval_metric` Set to "f","auc","tpr","fnr","fpr","tnr","prbe","f","odds"
- `TreeMethod` Choose from "hist", "gpu_hist"
- `MaxModelsInGrid` Number of models to test from grid options (243 total possible options)
- `NThreads` Set the maximum number of threads you'd like to dedicate to the model run. E.g. 8
- `model_path` A character string of your path file to where you want your output saved
- `ModelID` A character string to name your model and output
- `NumOfParDepPlots` Tell the function the number of partial dependence calibration plots you want to create.
- `Verbose` Set to 0 if you want to suppress model evaluation updates in training
- `ReturnModelObjects` Set to TRUE to output all modeling objects (E.g. plots and evaluation metrics)
- `SaveModelObjects` Set to TRUE to return all modeling objects to your environment
- `PassInGrid` Default is NULL. Provide a data.table of grid options from a previous run.

Value

Saves to file and returned in list: VariableImportance.csv, Model, ValidationData.csv, EvalutionPlot.png, EvaluationMetrics.csv, ParDepPlots.R a named list of features with partial dependence calibration plots, GridCollect, and GridList

Author(s)

Adrian Antico

See Also

Other Supervised Learning: AutoCatBoostClassifier, AutoCatBoostMultiClass, AutoCatBoostRegression, AutoCatBoostScoring, AutoH2OMLScoring, AutoH2OModeler, AutoH2OScoring, AutoH2oDRFClassifier, AutoH2oDRFMultiClass, AutoH2oDRFRegression, AutoH2oGBMClassifier, AutoH2oGBMMultiClass, AutoH2oGBMRegression, AutoMLTS, AutoNLS, AutoRecommenderScoring, AutoRecommender, AutoTS, AutoXGBoostMultiClass, AutoXGBoostRegression, AutoXGBoostScoring

Examples

```
Correl <- 0.85
N <- 10000
data <- data.table::data.table(Target = runif(N))
data[, x1 := qnorm(Target)]
data[, x2 := runif(N)]
data[, Independent_Variable1 := log(pnorm(Correl * x1 +
```

```
                                           sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable2 := (pnorm(Correl * x1 +
                                        sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable3 := exp(pnorm(Correl * x1 +
                                           sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                               sqrt(1-Correl^2) * qnorm(x2))))]
data[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable6 := (pnorm(Correl * x1 +
                                        sqrt(1-Correl^2) * qnorm(x2)))^0.10]
data[, Independent_Variable7 := (pnorm(Correl * x1 +
                                        sqrt(1-Correl^2) * qnorm(x2)))^0.25]
data[, Independent_Variable8 := (pnorm(Correl * x1 +
                                        sqrt(1-Correl^2) * qnorm(x2)))^0.75]
data[, Independent_Variable9 := (pnorm(Correl * x1 +
                                        sqrt(1-Correl^2) * qnorm(x2)))^2]
data[, Independent_Variable10 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^4]
data[, Independent_Variable11 := as.factor(
  ifelse(Independent_Variable2 < 0.20, "A",
         ifelse(Independent_Variable2 < 0.40, "B",
                ifelse(Independent_Variable2 < 0.6,  "C",
                       ifelse(Independent_Variable2 < 0.8,  "D", "E")))))]
data[, ':=' (x1 = NULL, x2 = NULL)]
data[, Target := ifelse(Target > 0.5, 1, 0)]
TestModel <- AutoXGBoostClassifier(data,
                                   ValidationData = NULL,
                                   TestData = NULL,
                                   TargetColumnName = 1,
                                   FeatureColNames = 2:12,
                                   IDcols = NULL,
                                   eval_metric = "auc",
                                   Trees = 50,
                                   GridTune = TRUE,
                                   grid_eval_metric = "auc",
                                   MaxModelsInGrid = 10,
                                   NThreads = 8,
                                   TreeMethod = "hist",
                                   model_path = getwd(),
                                   ModelID = "FirstModel",
                                   NumOfParDepPlots = 3,
                                   ReturnModelObjects = TRUE,
                                   SaveModelObjects = FALSE,
                                   PassInGrid = NULL)
```

`AutoXGBoostMultiClass` *AutoXGBoostMultiClass is an automated XGBoost modeling framework with grid-tuning and model evaluation*

Description

AutoXGBoostMultiClass is an automated XGBoost modeling framework with grid-tuning and model evaluation that runs a variety of steps. First, a stratified sampling (by the target variable) is done to create train and validation sets. Then, the function will run a random grid tune over N number of models and find which model is the best (a default model is always included in that set). Once the model is identified and built, several other outputs are generated: validation data with predictions, evaluation metrics, variable importance, and column names used in model fitting.

Usage

```
AutoXGBoostMultiClass(data, ValidationData = NULL, TestData = NULL,
  TargetColumnName = NULL, FeatureColNames = NULL, IDcols = NULL,
  eval_metric = "merror", Trees = 50, GridTune = FALSE,
  grid_eval_metric = "merror", TreeMethod = "hist",
  MaxModelsInGrid = 10, NThreads = 8, model_path = NULL,
  ModelID = "FirstModel", Verbose = 0, ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE, PassInGrid = NULL)
```

Arguments

`data`	This is your data set for training and testing your model
`ValidationData`	This is your holdout data set used in modeling either refine your hyperparameters.
`TestData`	This is your holdout data set. Catboost using both training and validation data in the training process so you should evaluate out of sample performance with this data set.
`TargetColumnName`	Either supply the target column name OR the column number where the target is located (but not mixed types). Target should be in factor or character form.
`FeatureColNames`	Either supply the feature column names OR the column number where the target is located (but not mixed types)
`IDcols`	A vector of column names or column numbers to keep in your data but not include in the modeling.
`eval_metric`	This is the metric used to identify best grid tuned model. Choose from "merror", "mlogloss"
`Trees`	The maximum number of trees you want in your models
`GridTune`	Set to TRUE to run a grid tuning procedure. Set a number in MaxModelsInGrid to tell the procedure how many models you want to test.
`grid_eval_metric`	Set to "accuracy" (only option currently)
`TreeMethod`	Choose from "hist", "gpu_hist"
`MaxModelsInGrid`	Number of models to test from grid options (243 total possible options)
`NThreads`	Set the maximum number of threads you'd like to dedicate to the model run. E.g. 8
`model_path`	A character string of your path file to where you want your output saved
`ModelID`	A character string to name your model and output
`Verbose`	Set to 0 if you want to suppress model evaluation updates in training

ReturnModelObjects
: Set to TRUE to output all modeling objects (E.g. plots and evaluation metrics)

SaveModelObjects
: Set to TRUE to return all modeling objects to your environment

PassInGrid
: Default is NULL. Provide a data.table of grid options from a previous run.

Value

Saves to file and returned in list: VariableImportance.csv, Model, ValidationData.csv, EvaluationMetrics.csv, GridCollect, and GridList

Author(s)

Adrian Antico

See Also

Other Supervised Learning: AutoCatBoostClassifier, AutoCatBoostMultiClass, AutoCatBoostRegression, AutoCatBoostScoring, AutoH2OMLScoring, AutoH2OModeler, AutoH2OScoring, AutoH2oDRFClassifier, AutoH2oDRFMultiClass, AutoH2oDRFRegression, AutoH2oGBMClassifier, AutoH2oGBMMultiClass, AutoH2oGBMRegression, AutoMLTS, AutoNLS, AutoRecommenderScoring, AutoRecommender, AutoTS, AutoXGBoostClassifier, AutoXGBoostRegression, AutoXGBoostScoring

Examples

```
Correl <- 0.85
N <- 10000
data <- data.table::data.table(Target = runif(N))
data[, x1 := qnorm(Target)]
data[, x2 := runif(N)]
data[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable2 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable3 := exp(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                                sqrt(1-Correl^2) * qnorm(x2))))]
data[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                             sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable6 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.10]
data[, Independent_Variable7 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.25]
data[, Independent_Variable8 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.75]
data[, Independent_Variable9 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^2]
data[, Independent_Variable10 := (pnorm(Correl * x1 +
                                          sqrt(1-Correl^2) * qnorm(x2)))^4]
data[, Target := as.factor(
  ifelse(Independent_Variable2 < 0.20, "A",
         ifelse(Independent_Variable2 < 0.40, "B",
```

```
                ifelse(Independent_Variable2 < 0.6,  "C",
                       ifelse(Independent_Variable2 < 0.8,  "D", "E")))))]
data[, Independent_Variable11 := as.factor(
ifelse(Independent_Variable2 < 0.25, "A",
       ifelse(Independent_Variable2 < 0.35, "B",
              ifelse(Independent_Variable2 < 0.65,  "C",
                     ifelse(Independent_Variable2 < 0.75,  "D", "E")))))]
data[, ':=' (x1 = NULL, x2 = NULL)]
TestModel <- AutoXGBoostMultiClass(data,
                                   ValidationData = NULL,
                                   TestData = NULL,
                                   TargetColumnName = 1,
                                   FeatureColNames = 2:12,
                                   IDcols = NULL,
                                   eval_metric = "merror",
                                   Trees = 50,
                                   GridTune = TRUE,
                                   grid_eval_metric = "accuracy",
                                   MaxModelsInGrid = 10,
                                   NThreads = 8,
                                   TreeMethod = "hist",
                                   model_path = getwd(),
                                   ModelID = "FirstModel",
                                   ReturnModelObjects = TRUE,
                                   SaveModelObjects = FALSE,
                                   PassInGrid = NULL)
```

`AutoXGBoostRegression` *AutoXGBoostRegression is an automated XGBoost modeling framework with grid-tuning and model evaluation*

Description

AutoXGBoostRegression is an automated XGBoost modeling framework with grid-tuning and model evaluation that runs a variety of steps. First, the function will run a random grid tune over N number of models and find which model is the best (a default model is always included in that set). Once the model is identified and built, several other outputs are generated: validation data with predictions, evaluation plot, evaluation boxplot, evaluation metrics, variable importance, partial dependence calibration plots, partial dependence calibration box plots, and column names used in model fitting.

Usage

```
AutoXGBoostRegression(data, ValidationData = NULL, TestData = NULL,
  TargetColumnName = NULL, FeatureColNames = NULL, IDcols = NULL,
  eval_metric = "RMSE", Trees = 50, GridTune = FALSE,
  grid_eval_metric = "mae", TreeMethod = "hist",
  MaxModelsInGrid = 10, NThreads = 8, model_path = NULL,
  ModelID = "FirstModel", NumOfParDepPlots = 3, Verbose = 0,
  ReturnModelObjects = TRUE, SaveModelObjects = FALSE,
  PassInGrid = NULL)
```

Arguments

- `data` This is your data set for training and testing your model
- `ValidationData` This is your holdout data set used in modeling either refine your hyperparameters.
- `TestData` This is your holdout data set. Catboost using both training and validation data in the training process so you should evaluate out of sample performance with this data set.
- `TargetColumnName` Either supply the target column name OR the column number where the target is located (but not mixed types).
- `FeatureColNames` Either supply the feature column names OR the column number where the target is located (but not mixed types)
- `IDcols` A vector of column names or column numbers to keep in your data but not include in the modeling.
- `eval_metric` This is the metric used to identify best grid tuned model. Choose from "r2", "RMSE", "MSE", "MAE"
- `Trees` The maximum number of trees you want in your models
- `GridTune` Set to TRUE to run a grid tuning procedure. Set a number in MaxModelsInGrid to tell the procedure how many models you want to test.
- `grid_eval_metric` Choose from "poisson","mae","mape","mse","msle","kl","cs","r2"
- `TreeMethod` Choose from "hist", "gpu_hist"
- `MaxModelsInGrid` Number of models to test from grid options (243 total possible options)
- `NThreads` Set the maximum number of threads you'd like to dedicate to the model run. E.g. 8
- `model_path` A character string of your path file to where you want your output saved
- `ModelID` A character string to name your model and output
- `NumOfParDepPlots` Tell the function the number of partial dependence calibration plots you want to create.
- `Verbose` Set to 0 if you want to suppress model evaluation updates in training
- `ReturnModelObjects` Set to TRUE to output all modeling objects (E.g. plots and evaluation metrics)
- `SaveModelObjects` Set to TRUE to return all modeling objects to your environment
- `PassInGrid` Default is NULL. Provide a data.table of grid options from a previous run.

Value

Saves to file and returned in list: VariableImportance.csv, Model, ValidationData.csv, EvalutionPlot.png, EvalutionBoxPlot.png, EvaluationMetrics.csv, ParDepPlots.R a named list of features with partial dependence calibration plots, ParDepBoxPlots.R, GridCollect, and GridList

Author(s)

Adrian Antico

See Also

Other Supervised Learning: AutoCatBoostClassifier, AutoCatBoostMultiClass, AutoCatBoostRegression, AutoCatBoostScoring, AutoH2OMLScoring, AutoH2OModeler, AutoH2OScoring, AutoH2oDRFClassifier, AutoH2oDRFMultiClass, AutoH2oDRFRegression, AutoH2oGBMClassifier, AutoH2oGBMMultiClass, AutoH2oGBMRegression, AutoMLTS, AutoNLS, AutoRecommenderScoring, AutoRecommender, AutoTS, AutoXGBoostClassifier, AutoXGBoostMultiClass, AutoXGBoostScoring

Examples

```
Correl <- 0.85
N <- 10000
data <- data.table::data.table(Target = runif(N))
data[, x1 := qnorm(Target)]
data[, x2 := runif(N)]
data[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable2 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable3 := exp(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                                sqrt(1-Correl^2) * qnorm(x2))))]
data[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                             sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable6 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.10]
data[, Independent_Variable7 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.25]
data[, Independent_Variable8 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^0.75]
data[, Independent_Variable9 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^2]
data[, Independent_Variable10 := (pnorm(Correl * x1 +
                                          sqrt(1-Correl^2) * qnorm(x2)))^4]
data[, Independent_Variable11 := as.factor(
  ifelse(Independent_Variable2 < 0.20, "A",
         ifelse(Independent_Variable2 < 0.40, "B",
                ifelse(Independent_Variable2 < 0.6,  "C",
                       ifelse(Independent_Variable2 < 0.8,  "D", "E")))))]
data[, ':=' (x1 = NULL, x2 = NULL)]
TestModel <- AutoXGBoostRegression(data,
                                   ValidationData = NULL,
                                   TestData = NULL,
                                   TargetColumnName = 1,
                                   FeatureColNames = 2:12,
                                   IDcols = NULL,
                                   eval_metric = "RMSE",
                                   Trees = 50,
                                   GridTune = TRUE,
                                   grid_eval_metric = "mae",
                                   MaxModelsInGrid = 10,
```

```
                                  NThreads = 8,
                                  TreeMethod = "hist",
                                  model_path = getwd(),
                                  ModelID = "FirstModel",
                                  NumOfParDepPlots = 3,
                                  ReturnModelObjects = TRUE,
                                  SaveModelObjects = FALSE,
                                  PassInGrid = NULL)
```

`AutoXGBoostScoring` *AutoXGBoostScoring is an automated scoring function that compliments the AutoCatBoost model training functions.*

Description

AutoXGBoostScoring is an automated scoring function that compliments the AutoCatBoost model training functions. This function requires you to supply features for scoring. It will run ModelDataPrep() and the DummifyDT() function to prepare your features for xgboost data conversion and scoring.

Usage

```
AutoXGBoostScoring(TargetType = NULL, ScoringData = NULL,
  FeatureColumnNames = NULL, IDcols = NULL, ModelPath = NULL,
  ModelID = NULL, ReturnFeatures = TRUE, MDP_Impute = TRUE,
  MDP_CharToFactor = TRUE, MDP_RemoveDates = TRUE,
  MDP_MissFactor = "0", MDP_MissNum = -1)
```

Arguments

`TargetType` Set this value to "regression", "classification", or "multiclass" to score models built using AutoCatBoostRegression(), AutoCatBoostClassify() or AutoCatBoostMultiClass().

`ScoringData` This is your data.table of features for scoring. Can be a single row or batch.

`FeatureColumnNames` Supply either column names or column numbers used in the AutoXGBoost__() function

`IDcols` Supply ID column numbers for any metadata you want returned with your predicted values

`ModelPath` Supply your path file used in the AutoXGBoost__() function

`ModelID` Supply the model ID used in the AutoXGBoost__() function

`ReturnFeatures` Set to TRUE to return your features with the predicted values.

`MDP_Impute` Set to TRUE if you did so for modeling and didn't do so before supplying ScoringData in this function

`MDP_CharToFactor` Set to TRUE to turn your character columns to factors if you didn't do so to your ScoringData that you are supplying to this function

MDP_RemoveDates
: Set to TRUE if you have date of timestamp columns in your ScoringData

MDP_MissFactor
: If you set MDP_Impute to TRUE, supply the character values to replace missing values with

MDP_MissNum
: If you set MDP_Impute to TRUE, supply a numeric value to replace missing values with

Value

A data.table of predicted values with the option to return model features as well.

See Also

Other Supervised Learning: AutoCatBoostClassifier, AutoCatBoostMultiClass, AutoCatBoostRegression, AutoCatBoostScoring, AutoH2OMLScoring, AutoH2OModeler, AutoH2OScoring, AutoH2oDRFClassifier, AutoH2oDRFMultiClass, AutoH2oDRFRegression, AutoH2oGBMClassifier, AutoH2oGBMMultiClass, AutoH2oGBMRegression, AutoMLTS, AutoNLS, AutoRecommenderScoring, AutoRecommender, AutoTS, AutoXGBoostClassifier, AutoXGBoostMultiClass, AutoXGBoostRegression

Examples

```
Preds <- AutoXGBoostScoring(TargetType = "regression",
                            ScoringData = data,
                            FeatureColumnNames = 2:12,
                            IDcols = NULL,
                            ModelPath = "home",
                            ModelID = "ModelTest",
                            ReturnFeatures = TRUE,
                            MDP_Impute = TRUE,
                            MDP_CharToFactor = TRUE,
                            MDP_RemoveDates = TRUE,
                            MDP_MissFactor = "0",
                            MDP_MissNum = -1)
```

ChartTheme *ChartTheme function is a ggplot theme generator for ggplots*

Description

This function helps your ggplots look professional with the choice of the two main colors that will dominate the theme

Usage

```
ChartTheme(Size = 12)
```

Arguments

Size
: The size of the axis labels and title

Value

An object to pass along to ggplot objects following the "+" sign

Author(s)

Adrian Antico

See Also

Other Misc: AutoH2OTextPrepScoring, AutoRecomDataCreate, PrintObjectsSize, RemixTheme, SimpleCap, multiplot, percRank, tempDatesFun, tokenizeH2O

Examples

```
data <- data.table::data.table(DateTime = as.Date(Sys.time()),
  Target = stats::filter(rnorm(1000,
                              mean = 50,
                              sd = 20),
                        filter=rep(1,10),
                        circular=TRUE))
data[, temp := seq(1:1000)][, DateTime := DateTime - temp][, temp := NULL]
data <- data[order(DateTime)]
p <- ggplot2::ggplot(data, ggplot2::aes(x = DateTime, y = Target)) + ggplot2::geom_line()
p <- p + ChartTheme(Size = 12)
```

CreateCalendarVariables *CreateCalendarVariables Create Caledar Variables*

Description

CreateCalendarVariables Rapidly creates calendar variables based on the date column you provide

Usage

```
CreateCalendarVariables(data, DateCols = c("Date", "Date2"),
  AsFactor = FALSE, TimeUnits = "wday")
```

Arguments

data	This is your data
DateCols	Supply either column names or column numbers of your date columns you want to use for creating calendar variables
AsFactor	Set to TRUE if you want factor type columns returned; otherwise integer type columns will be returned
TimeUnits	Supply a character vector of time units for creating calendar variables. Options include: "second", "minute", "hour", "wday", "mday", "yday", "week", "isoweek", "month", "quarter", "year"

Value

Returns your data.table with the added calendar variables at the end

Author(s)

Adrian Antico

See Also

Other Feature Engineering: AutoDataPartition, AutoWord2VecModeler, DT_GDL_Feature_Engineering, DummifyDT, FAST_GDL_Feature_Engineering, GDL_Feature_Engineering, ModelDataPrep, Scoring_GDL_Feature_E

Examples

```
data <- data.table::data.table(Date = "2018-01-01 00:00:00")
data <- CreateCalendarVariables(data,
                                DateCols = "Date",
                                AsFactor = FALSE,
                                TimeUnits = c("wday", "month", "year"))
```

DT_GDL_Feature_Engineering

An Automated Feature Engineering Function Using data.table frollmean

---

Description

Builds autoregressive and moving average from target columns and distributed lags and distributed moving average for independent features distributed across time. On top of that, you can also create time between instances along with their associated lags and moving averages. This function works for data with groups and without groups.

Usage

```
DT_GDL_Feature_Engineering(data, lags = c(seq(1, 50, 1)),
  periods = c(seq(5, 95, 5)), statsNames = c("MA"),
  targets = c("qty"), groupingVars = c("Group1", "Group2"),
  sortDateName = c("date"), timeDiffTarget = c("TimeDiffName"),
  timeAgg = c("days"), WindowingLag = 0, Type = c("Lag"),
  Timer = TRUE, SimpleImpute = TRUE)
```

Arguments

data	A data.table you want to run the function on
lags	A numeric vector of the specific lags you want to have generated. You must include 1 if WindowingLag = 1.
periods	A numeric vector of the specific rolling statistics window sizes you want to utilize in the calculations.
statsNames	A character vector of the corresponding names to create for the rollings stats variables.
targets	A character vector of the column names for the reference column in which you will build your lags and rolling stats
groupingVars	A character vector of categorical variable names you will build your lags and rolling stats by

`sortDateName` The column name of your date column used to sort events over time

`timeDiffTarget` Specify a desired name for features created for time between events. Set to NULL if you don't want time between events features created.

`timeAgg` List the time aggregation level for the time between events features, such as "hour", "day", "week", "month", "quarter", or "year"

`WindowingLag` Set to 0 to build rolling stats off of target columns directly or set to 1 to build the rolling stats off of the lag-1 target

`Type` List either "Lag" if you want features built on historical values or "Lead" if you want features built on future values

`Timer` Set to TRUE if you percentage complete tracker printout

`SimpleImpute` Set to TRUE for factor level imputation of "0" and numeric imputation of -1

Value

data.table of original data plus created lags, rolling stats, and time between event lags and rolling stats

Author(s)

Adrian Antico

See Also

Other Feature Engineering: AutoDataPartition, AutoWord2VecModeler, CreateCalendarVariables, DummifyDT, FAST_GDL_Feature_Engineering, GDL_Feature_Engineering, ModelDataPrep, Scoring_GDL_Feature_E

Examples

```
N = 25116
data <- data.table::data.table(DateTime = as.Date(Sys.time()),
                               Target = stats::filter(rnorm(N,
                                                            mean = 50,
                                                            sd = 20),
                                                      filter=rep(1,10),
                                                      circular=TRUE))
data[, temp := seq(1:N)][, DateTime := DateTime - temp][, temp := NULL]
data <- data[order(DateTime)]
data <- DT_GDL_Feature_Engineering(data,
                                   lags           = c(seq(1,5,1)),
                                   periods        = c(3,5,10,15,20,25),
                                   statsNames     = c("MA"),
                                   targets        = c("Target"),
                                   groupingVars   = NULL,
                                   sortDateName   = "DateTime",
                                   timeDiffTarget = c("Time_Gap"),
                                   timeAgg        = c("days"),
                                   WindowingLag   = 1,
                                   Type           = "Lag",
                                   Timer          = TRUE,
                                   SimpleImpute   = TRUE)
```

`DummifyDT` *DummifyDT creates dummy variables for the selected columns.*

Description

DummifyDT creates dummy variables for the selected columns. Either one-hot encoding, N+1 columns for N levels, or N columns for N levels.

Usage

```
DummifyDT(data, cols, KeepFactorCols = FALSE, OneHot = FALSE,
  SaveFactorLevels = FALSE, SavePath = NULL,
  ImportFactorLevels = FALSE, ClustScore = FALSE)
```

Arguments

`data`	The data set to run the micro auc on
`cols`	A vector with the names of the columns you wish to dichotomize
`KeepFactorCols`	Set to TRUE to keep the original columns used in the dichotomization process
`OneHot`	Set to TRUE to run one hot encoding, FALSE to generate N columns for N levels
`SaveFactorLevels`	Set to TRUE to save unique levels of each factor column to file as a csv
`SavePath`	Provide a file path to save your factor levels. Use this for models that you have to create dummy variables for.
`ImportFactorLevels`	Instead of using the data you provide, import the factor levels csv to ensure you build out all of the columns you trained with in modeling.
`ClustScore`	This is for scoring AutoKMeans. Set to FALSE for all other applications.

Value

data table with new dummy variables columns and optionally removes base columns

Author(s)

Adrian Antico

See Also

Other Feature Engineering: `AutoDataPartition`, `AutoWord2VecModeler`, `CreateCalendarVariables`, `DT_GDL_Feature_Engineering`, `FAST_GDL_Feature_Engineering`, `GDL_Feature_Engineering`, `ModelDataPrep`, `Scoring_GDL_Feature_Engineering`

Examples

```
test <- data.table::data.table(Value = runif(100000),
                   FactorCol = sample(x = c(letters,
                                            LETTERS,
                                            paste0(letters,letters),
                                            paste0(LETTERS,LETTERS),
                                            paste0(letters,LETTERS),
                                            paste0(LETTERS,letters)),
                                      size = 100000,
                                      replace = TRUE))
test <- DummifyDT(data = test,
                  cols = "FactorCol",
                  KeepFactorCols = FALSE)
ncol(test)
test[, sum(FactorCol_gg)]
```

EvalPlot — *EvalPlot automatically builds calibration plots for model evaluation*

Description

This function automatically builds calibration plots and calibration boxplots for model evaluation using regression, quantile regression, and binary and multinomial classification

Usage

```
EvalPlot(data, PredictionColName = c("PredictedValues"),
  TargetColName = c("ActualValues"), GraphType = c("calibration"),
  PercentileBucket = 0.05, aggrfun = function(x) mean(x, na.rm = TRUE))
```

Arguments

`data`	Data containing predicted values and actual values for comparison
`PredictionColName`	String representation of column name with predicted values from model
`TargetColName`	String representation of column name with target values from model
`GraphType`	Calibration or boxplot - calibration aggregated data based on summary statistic; boxplot shows variation
`PercentileBucket`	Number of buckets to partition the space on (0,1) for evaluation
`aggrfun`	The statistics function used in aggregation, listed as a function

Value

Calibration plot or boxplot

Author(s)

Adrian Antico

See Also

Other Model Evaluation and Interpretation: ParDepCalPlots, RedYellowGreen, threshOptim

Examples

```
Correl <- 0.85
data <- data.table::data.table(Target = runif(100))
data[, x1 := qnorm(Target)]
data[, x2 := runif(100)]
data[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                   sqrt(1-Correl^2) * qnorm(x2)))]
data[, Predict := (pnorm(Correl * x1 +
                         sqrt(1-Correl^2) * qnorm(x2)))]
EvalPlot(data,
         PredictionColName = "Predict",
         TargetColName = "Target",
         GraphType = "calibration",
         PercentileBucket = 0.05,
         aggrfun = function(x) quantile(x, probs = 0.50, na.rm = TRUE))
```

FAST_GDL_Feature_Engineering

An Fast Automated Feature Engineering Function

Description

For models with target variables within the realm of the current time frame but not too far back in time, this function creates autoregressive and rolling stats from target columns and distributed lags and distributed rolling stats for independent features distributed across time. On top of that, you can also create time between instances along with their associated lags and rolling stats. This function works for data with groups and without groups.

Usage

```
FAST_GDL_Feature_Engineering(data, lags = c(1:5), periods = c(seq(10,
  50, 10)), statsFUNs = c("mean", "median", "sd", "quantile85",
  "quantile95"), statsNames = c("mean", "median", "sd", "quantile85",
  "quantile95"), targets = c("Target"),
  groupingVars = c("GroupVariable"), sortDateName = c("DateTime"),
  timeDiffTarget = NULL, timeAgg = c("hours"), WindowingLag = 1,
  Type = c("Lag"), Timer = FALSE, SkipCols = FALSE,
  SimpleImpute = TRUE, AscRowByGroup = c("temp"), RecordsKeep = 1)
```

Arguments

data	A data.table you want to run the function on
lags	A numeric vector of the specific lags you want to have generated. You must include 1 if WindowingLag = 1.
periods	A numeric vector of the specific rolling statistics window sizes you want to utilize in the calculations.
statsFUNs	Vector of functions for your rolling windows, such as mean, sd, min, max, quantile

statsNames	A character vector of the corresponding names to create for the rollings stats variables.
targets	A character vector of the column names for the reference column in which you will build your lags and rolling stats
groupingVars	A character vector of categorical variable names you will build your lags and rolling stats by
sortDateName	The column name of your date column used to sort events over time
timeDiffTarget	Specify a desired name for features created for time between events. Set to NULL if you don't want time between events features created.
timeAgg	List the time aggregation level for the time between events features, such as "hour", "day", "week", "month", "quarter", or "year"
WindowingLag	Set to 0 to build rolling stats off of target columns directly or set to 1 to build the rolling stats off of the lag-1 target
Type	List either "Lag" if you want features built on historical values or "Lead" if you want features built on future values
Timer	Set to TRUE if you percentage complete tracker printout
SkipCols	Defaults to NULL; otherwise supply a character vector of the names of columns to skip
SimpleImpute	Set to TRUE for factor level imputation of "0" and numeric imputation of -1
AscRowByGroup	Required to have a column with a Row Number by group (if grouping) with 1 being the record for scoring (typically the most current in time)
RecordsKeep	List the number of records to retain (1 for last record, 2 for last 2 records, etc.)

Value

data.table of original data plus created lags, rolling stats, and time between event lags and rolling stats

Author(s)

Adrian Antico

See Also

Other Feature Engineering: AutoDataPartition, AutoWord2VecModeler, CreateCalendarVariables, DT_GDL_Feature_Engineering, DummifyDT, GDL_Feature_Engineering, ModelDataPrep, Scoring_GDL_Feature_Eng

Examples

```
N = 25116
data <- data.table::data.table(DateTime = as.Date(Sys.time()),
  Target = stats::filter(rnorm(N,
                               mean = 50,
                               sd = 20),
                         filter=rep(1,10),
                         circular=TRUE))
data[, temp := seq(1:N)][, DateTime := DateTime - temp]
data <- data[order(DateTime)]
data <- FAST_GDL_Feature_Engineering(data,
```

```
lags           = c(1:5),
periods        = c(seq(10,50,10)),
statsFUNs      = c("mean",
                   "median",
                   "sd",
                   "quantile85",
                   "quantile95"),
statsNames     = c("mean",
                   "median",
                   "sd",
                   "quantile85",
                   "quantile95"),
targets        = c("Target"),
groupingVars   = NULL,
sortDateName   = "DateTime",
timeDiffTarget = c("Time_Gap"),
timeAgg        = "days",
WindowingLag   = 1,
Type           = "Lag",
Timer          = TRUE,
SkipCols       = FALSE,
SimpleImpute   = TRUE,
AscRowByGroup  = "temp")
```

GDL_Feature_Engineering *An Automated Feature Engineering Function*

Description

Builds autoregressive and rolling stats from target columns and distributed lags and distributed rolling stats for independent features distributed across time. On top of that, you can also create time between instances along with their associated lags and rolling stats. This function works for data with groups and without groups.

Usage

```
GDL_Feature_Engineering(data, lags = c(seq(1, 5, 1)), periods = c(3, 5,
  10, 15, 20, 25), statsFUNs = c(function(x) quantile(x, probs = 0.1,
  na.rm = TRUE), function(x) quantile(x, probs = 0.9, na.rm = TRUE),
  function(x) base::mean(x, na.rm = TRUE), function(x) sd(x, na.rm = TRUE),
  function(x) quantile(x, probs = 0.25, na.rm = TRUE), function(x)
  quantile(x, probs = 0.75, na.rm = TRUE)), statsNames = c("q10", "q90",
  "mean", "sd", "q25", "q75"), targets = c("qty"),
  groupingVars = c("Group1", "Group2"), sortDateName = c("date"),
  timeDiffTarget = c("TimeDiffName"), timeAgg = c("days"),
  WindowingLag = 0, Type = c("Lag"), Timer = TRUE, SkipCols = NULL,
  SimpleImpute = TRUE)
```

Arguments

`data` A data.table you want to run the function on

`lags`	A numeric vector of the specific lags you want to have generated. You must include 1 if WindowingLag = 1.
`periods`	A numeric vector of the specific rolling statistics window sizes you want to utilize in the calculations.
`statsFUNs`	Vector that holds functions for your rolling stats, such as function(x) mean(x), function(x) sd(x), or function(x) quantile(x)
`statsNames`	A character vector of the corresponding names to create for the rollings stats variables.
`targets`	A character vector of the column names for the reference column in which you will build your lags and rolling stats
`groupingVars`	A character vector of categorical variable names you will build your lags and rolling stats by
`sortDateName`	The column name of your date column used to sort events over time
`timeDiffTarget`	Specify a desired name for features created for time between events. Set to NULL if you don't want time between events features created.
`timeAgg`	List the time aggregation level for the time between events features, such as "hour", "day", "week", "month", "quarter", or "year"
`WindowingLag`	Set to 0 to build rolling stats off of target columns directly or set to 1 to build the rolling stats off of the lag-1 target
`Type`	List either "Lag" if you want features built on historical values or "Lead" if you want features built on future values
`Timer`	Set to TRUE if you percentage complete tracker printout
`SkipCols`	Defaults to NULL; otherwise supply a character vector of the names of columns to skip
`SimpleImpute`	Set to TRUE for factor level imputation of "0" and numeric imputation of -1

Value

data.table of original data plus created lags, rolling stats, and time between event lags and rolling stats

Author(s)

Adrian Antico

See Also

Other Feature Engineering: `AutoDataPartition`, `AutoWord2VecModeler`, `CreateCalendarVariables`, `DT_GDL_Feature_Engineering`, `DummifyDT`, `FAST_GDL_Feature_Engineering`, `ModelDataPrep`, `Scoring_GDL_Feature_Engineering`

Examples

```
N = 25116
data <- data.table::data.table(DateTime = as.Date(Sys.time()),
  Target = stats::filter(rnorm(N,
                               mean = 50,
                               sd = 20),
                         filter=rep(1,10),
```

```
                          circular=TRUE))
data[, temp := seq(1:N)][, DateTime := DateTime - temp][, temp := NULL]
data <- data[order(DateTime)]
data <- GDL_Feature_Engineering(data,
           lags           = c(seq(1,1,1)),
           periods        = c(3),
           statsFUNs      = c(function(x) quantile(x, probs = 0.20, na.rm = TRUE)),
           statsNames     = c("q20"),
           targets        = c("Target"),
           groupingVars   = NULL,
           sortDateName   = "DateTime",
           timeDiffTarget = NULL,
           timeAgg        = "days",
           WindowingLag   = 1,
           Type           = "Lag",
           Timer          = TRUE,
           SkipCols       = FALSE,
           SimpleImpute   = TRUE)
```

`GenTSAnomVars` *GenTSAnomVars is an automated z-score anomaly detection via GLM-like procedure*

Description

GenTSAnomVars is an automated z-score anomaly detection via GLM-like procedure. Data is z-scaled and grouped by factors and time periods to determine which points are above and below the control limits in a cumulative time fashion. Then a cumulative rate is created as the final variable. Set KeepAllCols to FALSE to utilize the intermediate features to create rolling stats from them. The anomalies are separated into those that are extreme on the positive end versus those that are on the negative end.

Usage

```
GenTSAnomVars(data, ValueCol = "Value", GroupVar1 = "SKU",
  GroupVar2 = NULL, DateVar = "DATE", HighThreshold = 1.96,
  LowThreshold = -1.96, KeepAllCols = FALSE, IsDataScaled = TRUE)
```

Arguments

`data`	the source residuals data.table
`ValueCol`	the numeric column to run anomaly detection over
`GroupVar1`	this is a group by variable
`GroupVar2`	this is another group by variable
`DateVar`	this is a time variable for grouping
`HighThreshold`	this is the threshold on the high end
`LowThreshold`	this is the threshold on the low end
`KeepAllCols`	set to TRUE to remove the intermediate features
`IsDataScaled`	set to TRUE if you already scaled your data

Value

The original data.table with the added columns merged in. When KeepAllCols is set to FALSE, you will get back two columns: AnomHighRate and AnomLowRate - these are the cumulative anomaly rates over time for when you get anomalies from above the thresholds (e.g. 1.96) and below the thresholds.

Author(s)

Adrian Antico

See Also

Other Unsupervised Learning: AutoKMeans, ResidualOutliers

Examples

```
data <- data.table::data.table(DateTime = as.Date(Sys.time()),
  Target = stats::filter(rnorm(10000,
                               mean = 50,
                               sd = 20),
                         filter=rep(1,10),
                         circular=TRUE))
data[, temp := seq(1:10000)][, DateTime := DateTime - temp][, temp := NULL]
data <- data[order(DateTime)]
x <- data.table::as.data.table(sde::GBM(N=10000)*1000)
data[, predicted := x[-1,]]
stuff <- GenTSAnomVars(data,
                       ValueCol = "Target",
                       GroupVar1 = NULL,
                       GroupVar2 = NULL,
                       DateVar = "DateTime",
                       HighThreshold = 1.96,
                       LowThreshold = -1.96,
                       KeepAllCols = TRUE,
                       IsDataScaled  = FALSE)
```

ModelDataPrep *Final Data Preparation Function*

Description

This function replaces inf values with NA, converts characters to factors, and imputes with constants

Usage

```
ModelDataPrep(data, Impute = TRUE, CharToFactor = TRUE,
  RemoveDates = FALSE, MissFactor = "0", MissNum = -1,
  IgnoreCols = NULL)
```

Arguments

`data`	This is your source data you'd like to modify
`Impute`	Defaults to TRUE which tells the function to impute the data
`CharToFactor`	Defaults to TRUE which tells the function to convert characters to factors
`RemoveDates`	Defaults to FALSE. Set to TRUE to remove date columns from your data.table
`MissFactor`	Supply the value to impute missing factor levels
`MissNum`	Supply the value to impute missing numeric values
`IgnoreCols`	Supply column numbers for columns you want the function to ignore

Value

Returns the original data table with corrected values

Author(s)

Adrian Antico

See Also

Other Feature Engineering: `AutoDataPartition`, `AutoWord2VecModeler`, `CreateCalendarVariables`, `DT_GDL_Feature_Engineering`, `DummifyDT`, `FAST_GDL_Feature_Engineering`, `GDL_Feature_Engineering`, `Scoring_GDL_Feature_Engineering`

Examples

```
data <- data.table::data.table(Value = runif(100000),
                               FactorCol = as.character(sample(x = c(letters,
                                                                      LETTERS,
                                                                  paste0(letters,letters),
                                                                  paste0(LETTERS,LETTERS),
                                                                  paste0(letters,LETTERS),
                                                                 paste0(LETTERS,letters)),
                                                                 size = 100000,
                                                                 replace = TRUE)))
data <- ModelDataPrep(data,
                      Impute = TRUE,
                      CharToFactor = TRUE,
                      MissFactor = "0",
                      MissNum    = -1)
```

`multiplot` *Multiplot is a function for combining multiple plots*

Description

Sick of copying this one into your code? Well, not anymore.

Usage

```
multiplot(..., plotlist = NULL, cols = 2, layout = NULL)
```

Arguments

...	Passthrough arguments
plotlist	This is the list of your charts
cols	This is the number of columns in your multiplot
layout	Leave NULL

Value

Multiple ggplots on a single image

Author(s)

Adrian Antico

See Also

Other Misc: AutoH2OTextPrepScoring, AutoRecomDataCreate, ChartTheme, PrintObjectsSize, RemixTheme, SimpleCap, percRank, tempDatesFun, tokenizeH2O

Examples

```
Correl <- 0.85
data <- data.table::data.table(Target = runif(100))
data[, x1 := qnorm(Target)]
data[, x2 := runif(100)]
data[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Predict := (pnorm(Correl * x1 +
                           sqrt(1-Correl^2) * qnorm(x2)))]
p1 <- RemixAutoML::ParDepCalPlots(data,
                                  PredictionColName = "Predict",
                                  TargetColName = "Target",
                                  IndepVar = "Independent_Variable1",
                                  GraphType = "calibration",
                                  PercentileBucket = 0.20,
                                  FactLevels = 10,
                                  Function = function(x) mean(x, na.rm = TRUE))
p2 <- RemixAutoML::ParDepCalPlots(data,
                                  PredictionColName = "Predict",
                                  TargetColName = "Target",
                                  IndepVar = "Independent_Variable1",
                                  GraphType = "boxplot",
                                  PercentileBucket = 0.20,
                                  FactLevels = 10,
                                  Function = function(x) mean(x, na.rm = TRUE))
RemixAutoML::multiplot(plotlist = list(p1,p2), cols = 2)
```

`ParDepCalPlots` *ParDepCalPlots automatically builds partial dependence calibration plots for model evaluation*

Description

This function automatically builds partial dependence calibration plots and partial dependence calibration boxplots for model evaluation using regression, quantile regression, and binary and multinomial classification

Usage

```
ParDepCalPlots(data, PredictionColName = c("PredictedValues"),
  TargetColName = c("ActualValues"),
  IndepVar = c("Independent_Variable_Name"),
  GraphType = c("calibration"), PercentileBucket = 0.05,
  FactLevels = 10, Function = function(x) base::mean(x, na.rm = TRUE))
```

Arguments

`data`	Data containing predicted values and actual values for comparison
`PredictionColName`	Predicted values column names
`TargetColName`	Target value column names
`IndepVar`	Independent variable column names
`GraphType`	calibration or boxplot - calibration aggregated data based on summary statistic; boxplot shows variation
`PercentileBucket`	Number of buckets to partition the space on (0,1) for evaluation
`FactLevels`	The number of levels to show on the chart (1. Levels are chosen based on frequency; 2. all other levels grouped and labeled as "Other")
`Function`	Supply the function you wish to use for aggregation.

Value

Partial dependence calibration plot or boxplot

Author(s)

Adrian Antico

See Also

Other Model Evaluation and Interpretation: `EvalPlot`, `RedYellowGreen`, `threshOptim`

Examples

```
Correl <- 0.85
data <- data.table::data.table(Target = runif(100))
data[, x1 := qnorm(Target)]
data[, x2 := runif(100)]
data[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Predict := (pnorm(Correl * x1 +
                           sqrt(1-Correl^2) * qnorm(x2)))]
p1 <- RemixAutoML::ParDepCalPlots(data,
                                  PredictionColName = "Predict",
                                  TargetColName = "Target",
                                  IndepVar = "Independent_Variable1",
                                  GraphType = "calibration",
                                  PercentileBucket = 0.20,
                                  FactLevels = 10,
                                  Function = function(x) mean(x, na.rm = TRUE))
p1
```

---

percRank — *Percentile rank function*

Description

This function computes percentile ranks for each row in your data like Excel's PERCENT_RANK

Usage

```
percRank(x)
```

Arguments

x	X is your variable of interest

Value

vector of percentile ranks

Author(s)

Adrian Antico

See Also

Other Misc: AutoH2OTextPrepScoring, AutoRecomDataCreate, ChartTheme, PrintObjectsSize, RemixTheme, SimpleCap, multiplot, tempDatesFun, tokenizeH2O

Examples

```
data <- data.table::data.table(A = runif(100))
data[, Rank := percRank(A)]
data <- data.table::data.table(A = runif(100))
data[, Percentile := percRank(A)]
```

`PrintObjectsSize` *PrintObjectsSize prints out the top N objects and their associated sizes, sorted by size*

Description

PrintObjectsSize prints out the top N objects and their associated sizes, sorted by size

Usage

```
PrintObjectsSize(N = 10)
```

Arguments

N	The number of objects to display

Value

A print to your console of the sizes of the objects in your environment

Author(s)

Adrian Antico

See Also

Other Misc: AutoH2OTextPrepScoring, AutoRecomDataCreate, ChartTheme, RemixTheme, SimpleCap, multiplot, percRank, tempDatesFun, tokenizeH2O

Examples

```
PrintObjectsSize(N = 10)
```

`ProblematicFeatures` *ProblematicFeatures identifies problematic features for machine learning*

Description

ProblematicFeatures identifies problematic features for machine learning and outputs a data.table of the feature names in the first column and the metrics they failed to pass in the columns.

Usage

```
ProblematicFeatures(data, ColumnNumbers = c(1:ncol(data)),
  NearZeroVarThresh = 0.05, CharUniqThresh = 0.5, NA_Rate = 0.2,
  Zero_Rate = 0.2, HighSkewThresh = 10)
```

Arguments

- `data` — The data.table with the columns you wish to have analyzed
- `ColumnNumbers` — A vector with the column numbers you wish to analyze
- `NearZeroVarThresh` — Set to NULL to not run NearZeroVar(). Checks to see if the percentage of values in your numeric columns that are not constant are greater than the value you set here. If not, the feature is collects and returned with the percentage unique value.
- `CharUniqThresh` — Set to NULL to not run CharUniqthresh(). Checks to see if the percentage of unique levels / groups in your categorical feature is greater than the value you supply. If it is, the feature name is returned with the percentage unique value.
- `NA_Rate` — Set to NULL to not run NA_Rate(). Checks to see if the percentage of NA's in your features is greater than the value you supply. If it is, the feature name is returned with the percentage of NA values.
- `Zero_Rate` — Set to NULL to not run Zero_Rate(). Checks to see if the percentage of zero's in your features is greater than the value you supply. If it is, the feature name is returned with the percentage of zero values.
- `HighSkewThresh` — Set to NULL to not run HighSkew(). Checks for numeric columns whose ratio of the sum of the top 5th percentile of values to the bottom 95th percentile of values is greater than the value you supply. If true, the column name and value is returned.

Value

data table with new dummy variables columns and optionally removes base columns

Author(s)

Adrian Antico

See Also

Other EDA: AutoWordFreq, ProblematicRecords

Examples

```
test <- data.table::data.table(RandomNum = runif(1000))
test[, NearZeroVarEx := ifelse(runif(1000) > 0.99, runif(1), 1)]
test[, CharUniqueEx := as.factor(ifelse(RandomNum < 0.95, sample(letters, size = 1), "FFF"))]
test[, NA_RateEx := ifelse(RandomNum < 0.95, NA, "A")]
test[, ZeroRateEx := ifelse(RandomNum < 0.95, 0, runif(1))]
test[, HighSkewThreshEx := ifelse(RandomNum > 0.96, 100000, 1)]
ProblematicFeatures(test,
                    ColumnNumbers = 2:ncol(test),
                    NearZeroVarThresh = 0.05,
                    CharUniqThresh = 0.50,
                    NA_Rate = 0.20,
                    Zero_Rate = 0.20,
                    HighSkewThresh = 10)
```

ProblematicRecords *ProblematicRecords identifies problematic records for further investigation*

Description

ProblematicRecords identifies problematic records for further investigation and data.table with 3 additional columns at the beginning of the data.table: PredictedOutlier (0 = no outlier, 1 = outlier), predict (raw H2O predicted value from Isolation Forest), and mean_length (mean length of number of splits)

Usage

```
ProblematicRecords(data, ColumnNumbers = NULL, Threshold = 0.975,
  MaxMem = "28G", NThreads = -1, NTrees = 100,
  SampleRate = (sqrt(5) - 1)/2)
```

Arguments

`data`	The data.table with the columns you wish to have analyzed
`ColumnNumbers`	A vector with the column numbers you wish to analyze
`Threshold`	Quantile value to find the cutoff value for classifying outliers
`MaxMem`	Specify the amount of memory to allocate to H2O. E.g. "28G"
`NThreads`	Specify the number of threads (E.g. cores * 2)
`NTrees`	Specify the number of decision trees to build
`SampleRate`	Specify the row sample rate per tree

Value

A data.table

Author(s)

Adrian Antico

See Also

Other EDA: `AutoWordFreq`, `ProblematicFeatures`

Examples

```
 Correl <- 0.85
N <- 10000
data <- data.table::data.table(Target = runif(N))
data[, x1 := qnorm(Target)]
data[, x2 := runif(N)]
data[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                          sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable2 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))]
```

```
data[, Independent_Variable3 := exp(pnorm(Correl * x1 +
                                            sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                                sqrt(1-Correl^2) * qnorm(x2))))]
data[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                             sqrt(1-Correl^2) * qnorm(x2)))]
data[, Independent_Variable6 := (pnorm(Correl * x1 +
                                        sqrt(1-Correl^2) * qnorm(x2)))^0.10]
data[, Independent_Variable7 := (pnorm(Correl * x1 +
                                        sqrt(1-Correl^2) * qnorm(x2)))^0.25]
data[, Independent_Variable8 := (pnorm(Correl * x1 +
                                        sqrt(1-Correl^2) * qnorm(x2)))^0.75]
data[, Independent_Variable9 := (pnorm(Correl * x1 +
                                        sqrt(1-Correl^2) * qnorm(x2)))^2]
data[, Independent_Variable10 := (pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))^4]
data[, Target := as.factor(
 ifelse(Independent_Variable2 < 0.20, "A",
        ifelse(Independent_Variable2 < 0.40, "B",
               ifelse(Independent_Variable2 < 0.6,  "C",
                      ifelse(Independent_Variable2 < 0.8,  "D", "E")))))]
data[, Independent_Variable11 := as.factor(
 ifelse(Independent_Variable2 < 0.15, "A",
        ifelse(Independent_Variable2 < 0.45, "B",
               ifelse(Independent_Variable2 < 0.65,  "C",
                      ifelse(Independent_Variable2 < 0.85,  "D", "E")))))]
data[, ':=' (x1 = NULL, x2 = NULL)]
Outliers <- ProblematicRecords(data,
                               ColumnNumbers = NULL,
                               Threshold = 0.95,
                               MaxMem = "28G",
                               NThreads = -1)
```

`RedYellowGreen` *RedYellowGreen is for determining the optimal thresholds for binary classification when do-nothing is an option*

Description

This function will find the optimial thresholds for applying the main label and for finding the optimial range for doing nothing when you can quantity the cost of doing nothing

Usage

```
RedYellowGreen(data, PredictColNumber = 2, ActualColNumber = 1,
  TruePositiveCost = 0, TrueNegativeCost = 0,
  FalsePositiveCost = -10, FalseNegativeCost = -50, MidTierCost = -2,
  Cores = 8, Precision = 0.01, Boundaries = c(0.05, 0.75))
```

Arguments

`data` data is the data table with your predicted and actual values from a classification model

PredictColNumber
: The column number where the prediction variable is located (in binary form)

ActualColNumber
: The column number where the target variable is located

TruePositiveCost
: This is the utility for generating a true positive prediction

TrueNegativeCost
: This is the utility for generating a true negative prediction

FalsePositiveCost
: This is the cost of generating a false positive prediction

FalseNegativeCost
: This is the cost of generating a false negative prediction

MidTierCost
: This is the cost of doing nothing (or whatever it means to not classify in your case)

Cores
: Number of cores on your machine

Precision
: Set the decimal number to increment by between 0 and 1

Boundaries
: Supply a vector of two values c(lower bound, upper bound) where the first value is the smallest threshold you want to test and the second value is the largest value you want to test. Note, if your results are at the boundaries you supplied, you should extent the boundary that was reached until the values is within both revised boundaries.

Value

A data table with all evaluated strategies, parameters, and utilities, along with a 3d scatterplot of the results

Author(s)

Adrian Antico

See Also

Other Model Evaluation and Interpretation: EvalPlot, ParDepCalPlots, threshOptim

Examples

```
data <- data.table::data.table(Target = runif(10))
data[, x1 := qnorm(Target)]
data[, x2 := runif(10)]
data[, Predict := log(pnorm(0.85 * x1 +
                            sqrt(1-0.85^2) * qnorm(x2)))]
data[, ':=' (x1 = NULL, x2 = NULL)]
data <- RedYellowGreen(data,
                       PredictColNumber  = 2,
                       ActualColNumber   = 1,
                       TruePositiveCost  = 0,
                       TrueNegativeCost  = 0,
                       FalsePositiveCost = -1,
                       FalseNegativeCost = -2,
                       MidTierCost = -0.5,
                       Precision = 0.5,
```

```
                        Cores = 1,
                        Boundaries = c(0.05,0.75))
```

RemixTheme *RemixTheme function is a ggplot theme generator for ggplots*

Description

This function adds the Remix Theme to ggplots

Usage

```
RemixTheme()
```

Value

An object to pass along to ggplot objects following the "+" sign

Author(s)

Douglas Pestana

See Also

Other Misc: AutoH2OTextPrepScoring, AutoRecomDataCreate, ChartTheme, PrintObjectsSize, SimpleCap, multiplot, percRank, tempDatesFun, tokenizeH2O

Examples

```
data <- data.table::data.table(DateTime = as.Date(Sys.time()),
  Target = stats::filter(rnorm(1000,
                               mean = 50,
                               sd = 20),
                         filter=rep(1,10),
                         circular=TRUE))
data[, temp := seq(1:1000)][, DateTime := DateTime - temp][, temp := NULL]
data <- data[order(DateTime)]
p <- ggplot2::ggplot(data, ggplot2::aes(x = DateTime, y = Target)) + ggplot2::geom_line()
p <- p + RemixTheme()
```

ResidualOutliers *ResidualOutliers is an automated time series outlier detection function*

Description

ResidualOutliers is an automated time series outlier detection function that utilizes tsoutliers and auto.arima. It looks for five types of outliers: "AO" Additive outliter - a singular extreme outlier that surrounding values aren't affected by; "IO" Innovational outlier - Initial outlier with subsequent anomalous values; "LS" Level shift - An initial outlier with subsequent observations being shifted by some constant on average; "TC" Transient change - initial outlier with lingering effects that dissapate exponentially over time; "SLS" Seasonal level shift - similar to level shift but on a seasonal scale.

Usage

```
ResidualOutliers(data, DateColName = "DateTime",
  TargetColName = "Target", PredictedColName = NULL,
  TimeUnit = "day", maxN = 5, tstat = 2)
```

Arguments

`data`	the source residuals data.table
`DateColName`	The name of your data column to use in reference to the target variable
`TargetColName`	The name of your target variable column
`PredictedColName`	The name of your predicted value column. If you supply this, you will run anomaly detection of the difference between the target variable and your predicted value. If you leave PredictedColName NULL then you will run anomaly detection over the target variable.
`TimeUnit`	The time unit of your date column: hour, day, week, month, quarter, year
`maxN`	the largest lag or moving average (seasonal too) values for the arima fit
`tstat`	the t-stat value for tsoutliers

Value

A named list containing FullData = original data.table with outliers data and ARIMA_MODEL = the arima model.

Author(s)

Adrian Antico

See Also

Other Unsupervised Learning: AutoKMeans, GenTSAnomVars

Examples

```
data <- data.table::data.table(DateTime = as.Date(Sys.time()),
                               Target = as.numeric(stats::filter(rnorm(1000,
                                                                       mean = 50,
                                                                       sd = 20),
                                                                 filter=rep(1,10),
                                                                 circular=TRUE)))
data[, temp := seq(1:1000)][, DateTime := DateTime - temp][, temp := NULL]
data <- data[order(DateTime)]
data[, Predicted := as.numeric(stats::filter(rnorm(1000,
                                                   mean = 50,
                                                   sd = 20),
                                             filter=rep(1,10),
                                             circular=TRUE))]
stuff <- ResidualOutliers(data = data,
                          DateColName = "DateTime",
                          TargetColName = "Target",
                          PredictedColName = NULL,
                          TimeUnit = "day",
                          maxN = 5,
```

```
                          tstat = 4)
data     <- stuff[[1]]
model    <- stuff[[2]]
outliers <- data[type != "<NA>"]
```

Scoring_GDL_Feature_Engineering

An Automated Scoring Feature Engineering Function

Description

For scoring purposes (brings back a single row by group), this function creates autoregressive and rolling stats from target columns and distributed lags and distributed rolling stats for independent features distributed across time. On top of that, you can also create time between instances along with their associated lags and rolling stats. This function works for data with groups and without groups.

Usage

```
Scoring_GDL_Feature_Engineering(data, lags = c(seq(1, 5, 1)),
  periods = c(3, 5, 10, 15, 20, 25), statsNames = c("MA"),
  targets = c("Target"), groupingVars = NULL,
  sortDateName = c("DateTime"), timeDiffTarget = c("Time_Gap"),
  timeAgg = "days", WindowingLag = 1, Type = "Lag", Timer = TRUE,
  SimpleImpute = TRUE, AscRowByGroup = "temp", RecordsKeep = 1)
```

Arguments

`data`	A data.table you want to run the function on
`lags`	A numeric vector of the specific lags you want to have generated. You must include 1 if WindowingLag = 1.
`periods`	A numeric vector of the specific rolling statistics window sizes you want to utilize in the calculations.
`statsNames`	A character vector of the corresponding names to create for the rollings stats variables.
`targets`	A character vector of the column names for the reference column in which you will build your lags and rolling stats
`groupingVars`	A character vector of categorical variable names you will build your lags and rolling stats by
`sortDateName`	The column name of your date column used to sort events over time
`timeDiffTarget`	Specify a desired name for features created for time between events. Set to NULL if you don't want time between events features created.
`timeAgg`	List the time aggregation level for the time between events features, such as "hour", "day", "week", "month", "quarter", or "year"
`WindowingLag`	Set to 0 to build rolling stats off of target columns directly or set to 1 to build the rolling stats off of the lag-1 target
`Type`	List either "Lag" if you want features built on historical values or "Lead" if you want features built on future values

`Timer`	Set to TRUE if you percentage complete tracker printout
`SimpleImpute`	Set to TRUE for factor level imputation of "0" and numeric imputation of -1
`AscRowByGroup`	Required to have a column with a Row Number by group (if grouping) with 1 being the record for scoring (typically the most current in time)
`RecordsKeep`	List the number of records to retain (1 for last record, 2 for last 2 records, etc.)

Value

data.table of original data plus created lags, rolling stats, and time between event lags and rolling stats

Author(s)

Adrian Antico

See Also

Other Feature Engineering: AutoDataPartition, AutoWord2VecModeler, CreateCalendarVariables, DT_GDL_Feature_Engineering, DummifyDT, FAST_GDL_Feature_Engineering, GDL_Feature_Engineering, ModelDataPrep

Examples

```
N = 25116
data1 <- data.table::data.table(DateTime = as.Date(Sys.time()),
                                Target = stats::filter(rnorm(N,
                                                             mean = 50,
                                                             sd = 20),
                                                       filter=rep(1,10),
                                                       circular=TRUE))
data1[, temp := seq(1:N)][, DateTime := DateTime - temp]
data1 <- data1[order(DateTime)]
data1 <- Scoring_GDL_Feature_Engineering(data1,
                                         lags           = c(seq(1,5,1)),
                                         periods        = c(3,5,10,15,20,25),
                                         statsNames     = c("MA"),
                                         targets        = c("Target"),
                                         groupingVars   = NULL,
                                         sortDateName   = c("DateTime"),
                                         timeDiffTarget = c("Time_Gap"),
                                         timeAgg        = "days",
                                         WindowingLag   = 1,
                                         Type           = "Lag",
                                         Timer          = TRUE,
                                         SimpleImpute   = TRUE,
                                         AscRowByGroup  = "temp",
                                         RecordsKeep    = 1)
```

SimpleCap — *SimpleCap function is for capitalizing the first letter of words*

Description

SimpleCap function is for capitalizing the first letter of words (need I say more?)

Usage

```
SimpleCap(x)
```

Arguments

x	Column of interest

Value

An object to pass along to ggplot objects following the "+" sign

Author(s)

Adrian Antico

See Also

Other Misc: AutoH2OTextPrepScoring, AutoRecomDataCreate, ChartTheme, PrintObjectsSize, RemixTheme, multiplot, percRank, tempDatesFun, tokenizeH2O

Examples

```
x <- "adrian"
x <- SimpleCap(x)
```

tempDatesFun — *tempDatesFun Convert Excel datetime char columns to Date columns*

Description

tempDatesFun takes the Excel datetime column, which imports as character, and converts it into a date type

Usage

```
tempDatesFun(x)
```

Arguments

x	The column of interest

Value

An object to pass along to ggplot objects following the "+" sign

Author(s)

Adrian Antico

See Also

Other Misc: AutoH2OTextPrepScoring, AutoRecomDataCreate, ChartTheme, PrintObjectsSize, RemixTheme, SimpleCap, multiplot, percRank, tokenizeH2O

Examples

```
Cdata <- data.table::data.table(DAY_DATE = "2018-01-01 8:53")
Cdata[, DAY_DATE := tempDatesFun(DAY_DATE)]
```

---

threshOptim *Utility maximizing thresholds for binary classification*

Description

This function will return the utility maximizing threshold for future predictions along with the data generated to estimate the threshold

Usage

```
threshOptim(data, actTar = "target", predTar = "p1", tpProfit = 0,
  tnProfit = 0, fpProfit = -1, fnProfit = -2)
```

Arguments

data	data is the data table you are building the modeling on
actTar	The column name where the actual target variable is located (in binary form)
predTar	The column name where the predicted values are located
tpProfit	This is the utility for generating a true positive prediction
tnProfit	This is the utility for generating a true negative prediction
fpProfit	This is the cost of generating a false positive prediction
fnProfit	This is the cost of generating a false negative prediction

Value

Optimal threshold and corresponding utilities for the range of thresholds tested

Author(s)

Adrian Antico

See Also

Other Model Evaluation and Interpretation: EvalPlot, ParDepCalPlots, RedYellowGreen

Examples

```
data <- data.table::data.table(Target = runif(10))
data[, x1 := qnorm(Target)]
data[, x2 := runif(10)]
data[, Predict := log(pnorm(0.85 * x1 +
                              sqrt(1-0.85^2) * qnorm(x2)))]
data[, ':=' (x1 = NULL, x2 = NULL)]
data <- threshOptim(data     = data,
                    actTar   = "Target",
                    predTar  = "Predict",
                    tpProfit = 0,
                    tnProfit = 0,
                    fpProfit = -1,
                    fnProfit = -2)
optimalThreshold <- data$Thresholds
allResults <- data$EvaluationTable
```

`tokenizeH2O` *For NLP work*

Description

This function tokenizes text data

Usage

```
tokenizeH2O(data)
```

Arguments

`data`	The text data

Author(s)

Adrian Antico

See Also

Other Misc: `AutoH2OTextPrepScoring`, `AutoRecomDataCreate`, `ChartTheme`, `PrintObjectsSize`, `RemixTheme`, `SimpleCap`, `multiplot`, `percRank`, `tempDatesFun`

Examples

```
data <- tokenizeH2O(data = data[["StringColumn"]])
```

Index