Package 'RemixAutoML'

December 1, 2020

Title Remix Automated Machine Learning

Version 0.2.7

Date 2020-11-04

Maintainer Adrian Antico <adrianantico@gmail.com>

Description R package for the automation of machine learning, forecasting, feature engineering, model evaluation, model interpretation, data generation, and recommenders. Build using data.table for all tabular data-related tasks.

License MPL-2.0 | file LICENSE

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

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

Depends R (>= 3.5.0)

Imports arules, bit64, catboost, combinat, data.table, doParallel, e1071, fBasics, foreach, forecast, ggplot2, grid, h2o, itertools, lime, lubridate, methods, MLmetrics, monreg, nortest, parallel, pROC, RColorBrewer, recommenderlab, scatterplot3d, stats, stringr, timeDate, tsoutliers, wordcloud, xgboost

Suggests knitr, rmarkdown, sde, testthat, fpp, gridExtra

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 7.1.1

SystemRequirements Java (>= 7.0)

Author Adrian Antico [aut, cre], Douglas Pestana [ctb]

ByteCompile TRUE

R topics documented:

`RemixAutoML-package` *Automated Machine Learning Remixed*

Description

Automated Machine Learning Remixed for real-world use-cases. The package utilizes data.table under the hood for all data wrangling like operations so it's super fast and memory efficient. All ML methods are available in R or Python. The forecasting functions are unique and state of the art. There are feature engineering functions in this package that you cannot find anywhere else.

Details

See the github README for details and examples www.github.com/AdrianAntico/RemixAutoML

Author(s)

Adrian Antico, adrianantico@gmail.com, Douglas Pestana

`AutoBanditNNet` *AutoBanditNNet*

Description

AutoBanditNNet is a multi-armed bandit model testing framework for AR and SAR NNets. Randomized probability matching is the underlying bandit algorithm. Model evaluation is done by blending the training error and the validation error from testing the model on out of sample data. The bandit algorithm compares the performance of the current build against the previous builds which starts with the classic nnetar model from the forecast package. Depending on how many lags, seasonal lags, and fourier pairs you test the number of combinations of features to test begins to approach 10,000 different combinations of settings. The function tests out transformations, differencing, and variations of the lags, seasonal lags, and fourier pairs. The paramter space is broken up into various buckets that are increasing in sophistication. The bandit algorithm samples from those buckets and based on many rounds of testing it determines which buckets to generate samples from more frequently based on the models performance coming from that bucket. All of the models have performance data collected on them and a final rebuild is initiated when a winner is found. The rebuild process begins by retraining the model with the settings that produced the best performance. If the model fails to build, for whatever reason, the next best buildable model is rebuilt.

Usage

```
AutoBanditNNet(
  data,
  TargetVariableName,
  DateColumnName,
  TimeAggLevel = "week",
  EvaluationMetric = "MAE",
  NumHoldOutPeriods = 5L,
  NumFCPeriods = 5L,
  MaxLags = 5L,
  MaxSeasonalLags = 1L,
  MaxFourierPairs = 2L,
  TrainWeighting = 0.5,
  MaxConsecutiveFails = 12L,
  MaxNumberModels = 100L,
  MaxRunTimeMinutes = 10L
)
```

Arguments

`data` Source data.table

`TargetVariableName` Name of your time series target variable

`DateColumnName` Name of your date column

`TimeAggLevel` Choose from "year", "quarter", "month", "week", "day", "hour"

`EvaluationMetric` Choose from MAE, MSE, and MAPE

`NumHoldOutPeriods` Number of time periods to use in the out of sample testing

`NumFCPeriods` Number of periods to forecast

`MaxLags` A single value of the max number of lags to test

`MaxSeasonalLags` A single value of the max number of seasonal lags to test

`MaxFourierPairs` A single value of the max number of fourier pairs to test

`TrainWeighting` Model ranking is based on a weighted average of training metrics and out of sample metrics. Supply the weight of the training metrics, such as 0.50 for 50 percent.

`MaxConsecutiveFails` When a new best model is found MaxConsecutiveFails resets to zero. Indicated the number of model attemps without a new winner before terminating the procedure.

`MaxNumberModels` Indicate the maximum number of models to test.

`MaxRunTimeMinutes` Indicate the maximum number of minutes to wait for a result.

Author(s)

Adrian Antico

See Also

Other Automated Time Series: AutoBanditSarima(), AutoCatBoostFreqSizeScoring(), AutoH2oGBMFreqSizeScor
AutoTBATS(), AutoTS()

AutoBanditSarima *AutoBanditSarima*

Description

AutoBanditSarima is a multi-armed bandit model testing framework for SARIMA. Randomized probability matching is the underlying bandit algorithm. Model evaluation is done by blending the training error and the validation error from testing the model on out of sample data. The bandit algorithm compares the performance of the current build against the previous builds which starts with the classic auto.arima from the forecast package. Depending on how many lags, moving averages, seasonal lags and moving averages you test the number of combinations of features to test begins to approach 100,000 different combinations of settings. The function tests out transformations, differencing, and variations of the lags and moving averages. The paramter space is broken up into various buckets that are increasing in sophistication. The bandit algorithm samples from those buckets and based on many rounds of testing it determines which buckets to generate samples from more frequently based on the models performance coming from that bucket. All of the models have performance data collected on them and a final rebuild is initiated when a winner is found. The rebuild process begins by retraining the model with the settings that produced the best performance. If the model fails to build, for whatever reason, the next best buildable model is rebuilt.

Usage

```
AutoBanditSarima(
  data,
  ByDataType = TRUE,
  TargetVariableName,
  DateColumnName,
  TimeAggLevel = "week",
  EvaluationMetric = "MAE",
  NumHoldOutPeriods = 5L,
  NumFCPeriods = 5L,
  MaxLags = 5L,
  MaxSeasonalLags = 0L,
  MaxMovingAverages = 5L,
  MaxSeasonalMovingAverages = 0L,
  MaxFourierPairs = 2L,
  TrainWeighting = 0.5,
  MaxConsecutiveFails = 25L,
  MaxNumberModels = 100L,
  MaxRunTimeMinutes = 10L,
  NumberCores = max(1L, parallel::detectCores()),
  DebugMode = FALSE
)
```

Arguments

`data`	Source data.table
`ByDataType`	TRUE returns the best model from the four base sets of possible models. FALSE returns the best model.
`TargetVariableName`	Name of your time series target variable
`DateColumnName`	Name of your date column
`TimeAggLevel`	Choose from "year", "quarter", "month", "week", "day", "hour"
`EvaluationMetric`	Choose from MAE, MSE, and MAPE
`NumHoldOutPeriods`	Number of time periods to use in the out of sample testing
`NumFCPeriods`	Number of periods to forecast
`MaxLags`	A single value of the max number of lags to test
`MaxSeasonalLags`	A single value of the max number of seasonal lags to test
`MaxMovingAverages`	A single value of the max number of moving averages to test
`MaxSeasonalMovingAverages`	A single value of the max number of seasonal moving averages to test
`MaxFourierPairs`	A single value of the max number of fourier pairs to test
`TrainWeighting`	Model ranking is based on a weighted average of training metrics and out of sample metrics. Supply the weight of the training metrics, such as 0.50 for 50 percent.
`MaxConsecutiveFails`	When a new best model is found MaxConsecutiveFails resets to zero. Indicated the number of model attemps without a new winner before terminating the procedure.
`MaxNumberModels`	Indicate the maximum number of models to test.
`MaxRunTimeMinutes`	Indicate the maximum number of minutes to wait for a result.
`NumberCores`	Number of cores to use in parallelism. E.g. if you have a 4 core CPU then supply 4 if you want to utilize all four cores
`DebugMode`	Set to TRUE to get print outs of particular steps helpful in tracing errors

Value

data.table containing historical values and the forecast values along with the grid tuning results in full detail, as a second data.table

Author(s)

Adrian Antico

See Also

Other Automated Time Series: `AutoBanditNNet()`, `AutoCatBoostFreqSizeScoring()`, `AutoH2oGBMFreqSizeScoring`
`AutoTBATS()`, `AutoTS()`

Examples

```
## Not run:
# Build model
data <- RemixAutoML::FakeDataGenerator(
  TimeSeries = TRUE, TimeSeriesTimeAgg = "1min")

# Pimping
Output <- RemixAutoML::AutoBanditSarima(
  data = data,
  ByDataType = FALSE,
  TargetVariableName = "Weekly_Sales",
  DateColumnName = "Date",
  TimeAggLevel = "1min",
  EvaluationMetric = "MAE",
  NumHoldOutPeriods = 12L,
  NumFCPeriods = 16L,
  MaxLags = 10L,
  MaxSeasonalLags = 0L,
  MaxMovingAverages = 3L,
  MaxSeasonalMovingAverages = 0L,
  MaxFourierPairs = 2L,
  TrainWeighting = 0.50,
  MaxConsecutiveFails = 50L,
  MaxNumberModels = 100L,
  MaxRunTimeMinutes = 10L,
  NumberCores = 12,
  DebugMode = FALSE)

# Output
Output$ForecastPlot
Output$Forecast
Output$PerformanceGrid
Output$ErrorLagMA2x2

## End(Not run)
```

AutoCARMA_QA *AutoCARMA_QA*

Description

AutoCARMA_QA

Usage

```
AutoCARMA_QA(
  ModelName = "catboost",
  FeatureGridTune = FALSE,
  MaxMem_ = "28G",
  NThreads_ = max(1, parallel::detectCores() - 2),
  TreeMethod__ = "hist",
  TestRows = "ALL",
  DataPath = "C:/Users/Bizon/Documents/GitHub/QA_DataSets",
```

```
  dataForecastX = "CARMA-WALMART-2GroupVars_FC.csv",
  dataX = "OneGroup-Eval-Walmart.csv",
  XREGSX = "CARMA-WALMART-2GroupVars-XREGS_2Var.csv",
  TargetColumnName_ = "Weekly_Sales",
  DateColumnName_ = "Date",
  HierarchGroups_ = c("Store", "Dept"),
  GroupVariables_ = c("Store", "Dept"),
  TimeUnit_ = "week",
  TimeGroups_ = c("week", "month", "quarter"),
  ZeroPadSeries_ = NULL,
  DataTruncate_ = FALSE,
  SplitRatios_ = c(1 - 3/143, 3/143),
  PartitionType_ = "timeseries",
  TrainOnFull_ = FALSE,
  FC_Periods_ = 4,
  EvalMetric_ = "RMSE",
  GridTune_ = FALSE,
  GridEvalMetric_ = "mae",
  ModelCount_ = 5,
  TaskType_ = "GPU",
  Timer_ = TRUE,
  TargetTransformation_ = TRUE,
  Difference_ = TRUE,
  CalendarVariables_ = TRUE,
  HolidayVariable_ = TRUE,
  HolidayLags_ = 1,
  HolidayMovingAverages_ = 1:2,
  Lags_ = c(1:5),
  MA_Periods_ = c(1:5),
  SD_Periods_ = c(2:5),
  Skew_Periods_ = c(3:5),
  Kurt_Periods_ = c(4:5),
  Quantile_Periods_ = c(3:5),
  Quantiles_Selected_ = c("q5", "q95"),
  FourierTerms_ = 4,
  TimeTrendVariable_ = TRUE,
  NTrees_ = 150,
  DebugMode_ = TRUE,
  OptionsWarn = 1
)
```

Arguments

- `ModelName` Choose from 'catboost', 'h2odrf', 'h2ogbm', 'h2oglm', 'h2oautoml', 'xgboost'
- `FeatureGridTune` Set to TRUE to only run in evaluation model opposed to TrainOnFull model which does not return model performance measures
- `MaxMem_` = "28G"
- `NThreads_` = parallel::detectCores() - 2
- `TreeMethod__` = "hist" or "gpu_hist" for xgboost carma
- `TestRows` = "ALL" to run all tests (see example for all tests), or a numeric vector with the row numbers from the test list (see example)

DataPath In quotes, provide the file path to where your data is stored

dataForecastX = "RawDataXREG.csv" Use quotes. # Be aware that grouped data and using XREGS_ requires that your joining group variables have the same name. MUST SUPPLY VALUE

dataX = "RawDataXREG.csv" Use quotes. # Be aware that grouped data and using XREGS_ requires that your joining group variables have the same name. MUST SUPPLY VALUE

XREGSX = "XREG.csv" Use quotes. # data.table with ONLY 3 COLUMN TYPES: 1: - GroupVariables_ and DateColumnName_ join-by variables with matching join column names and data types compared to data_ and; 2 - features - needs to exist for all historical periods matching data_ along with a sufficient amount of data to cover the forecast period as defined by FC_Periods_. OR Supply NULL to arg.

TargetColumnName_
= "Weekly_Sales" # WalmartData target column name.

DateColumnName_
= "Date" # Name of data_ date column name.

HierarchGroups_
= c("Store","Dept") # NULL otherwise

GroupVariables_
= c("Store","Dept") #

TimeUnit_ = "week" # Choices include "1min", "5min", "10min", "15min", "30min", "hour", "day", "week", "month", "quarter", "year"

TimeGroups_ = c("weeks","months","quarter") # These will tell GDL to build gdl features along the time aggregation dimension

ZeroPadSeries_ = c('NULL', 'all', 'inner') ZeroPadSeries choose "all", "inner", or NULL. 'Outer' grows missing dates by group to the largest of all groups size. 'Inner' fills in series by using the group level's own max and min values (versus filling all group levels to the max value of the groups level with the widest time gap)

DataTruncate_ = FALSE # TRUE will truncate all rows where GDL columns produced a -1 (remove all rows where ID < max(rolling stats)). FALSE otherwise.

SplitRatios_ = c(1 - 10 / 143, 10 / 143) # If you have GroupVariables_ then base it on number of records in a group, like default

PartitionType_ = "timeseries" # always time series for this function. Place holder for other time series options down the road.

TrainOnFull_ = FALSE # Set to TRUE put in Forecase mode. FALSE to put in Evaluation mode. Forecast mode generates forecasts based on a model built using all of data_, and no evaluation metrics are collected when set to TRUE. Evaluation mode will build a forecast for your validation periods and collect the holdout metrics and other evaluation objects, but no future forecast beyond max date of data_. as specified in SplitRatios_.

FC_Periods_ = 4 # Self explanatory

EvalMetric_ = "RMSE" # "RMSE" only with catboost 17.5

GridTune_ = FALSE # NEEDS TO BE UPDATED ONCE BANDIT GRID TUNING WORKS.

GridEvalMetric_
= "mae" # 'poisson', 'mae', 'mape', 'mse', 'msle', 'kl', 'cs', 'r2'. If metric computation fails then no output is generated in final metric evaluation data.table

ModelCount_ = 5 # NEEDS TO BE UPDATED ONCE BANDIT GRID TUNING WORKS.

TaskType_ = "GPU" # Set to "CPU" to train on CPU versus GPU. Must supply a value.

Timer_ = TRUE # Print out the forecast step the function is currently working on. If it errors on the first run scoring the model then it is likely a very different error then if has printed "Forecasting 1:"

TargetTransformation_
= TRUE # Set to TRUE to have every available numeric transformation compete for best normalization fit to normal distribution

Difference_ = TRUE # The I in ARIMA. Works for single series and grouped series a.k.a. panel data.

CalendarVariables_
= TRUE # This TURNS ON procedure to create numeric calendar variables that your TimeUnit_ directs. FALSE otherwise.

HolidayVariable_
= TRUE # This TURNS ON procedure to create a numeric holiday count variable. FALSE otherwise.

HolidayLags_ = c(1:2) # Supply a numeric vector of lag periods

HolidayMovingAverages_
= c(1:2) # Supply a numeric vector of Moving Average periods

Lags_ = c(1:5) # Numeric vector of lag periods

MA_Periods_ = c(1:5) # Numeric vector of lag periods

SD_Periods_ = c(2:5) # Numeric vector of lag periods

Skew_Periods_ = c(3:5) # Numeric vector of lag periods

Kurt_Periods_ = c(4:5) # Numeric vector of lag periods

Quantile_Periods_
= c(3:5) # Numeric vector of lag periods

Quantiles_Selected_
= c("q5","q95") # Select the quantiles you want calculated. "q5", "q10", ..., "q95".

FourierTerms_ = 2 # (TECHINICALLY FOURIER PAIRS) Hierarchy grouping (full group variable interaction set) is ran by default (MAKE INTO OPTIOn). Uses parallelization to loop through the unique set of all GroupVariables levels and computes fourier terms as if the group level's are a single series; just for all groups and it's parallelized.

TimeTrendVariable_
= TRUE # Set to TRUE to have a sequence created from 1 to nrow by group or single series

NTrees_ = 150 # Number of trees to have trained. Can be 10000 or more depending on group level size.

DebugMode_ = TRUE # When TRUE it will print every comment section header line. When it crashes, you can get a print out of the last N steps that were ran, depending on the print max limit.

OptionsWarn Set to 1 to print warnings immediately to screen versus after a function finishes; 2 to kill processes if a warning occurs. See options(warn =)

Author(s)

Adrian Antico

AutoCatBoostCARMA *AutoCatBoostCARMA*

Description

AutoCatBoostCARMA Mutlivariate Forecasting with calendar variables, Holiday counts, holiday lags, holiday moving averages, differencing, transformations, interaction-based categorical encoding using target variable and features to generate various time-based aggregated lags, moving averages, moving standard deviations, moving skewness, moving kurtosis, moving quantiles, parallelized interaction-based fourier pairs by grouping variables, and Trend Variables.

Usage

```
AutoCatBoostCARMA(
  data,
  TimeWeights = NULL,
  NonNegativePred = FALSE,
  RoundPreds = FALSE,
  TrainOnFull = FALSE,
  TargetColumnName = "Target",
  DateColumnName = "DateTime",
  HierarchGroups = NULL,
  GroupVariables = NULL,
  FC_Periods = 30,
  TimeUnit = "week",
  TimeGroups = c("weeks", "months"),
  PDFOutputPath = NULL,
  SaveDataPath = NULL,
  NumOfParDepPlots = 10L,
  TargetTransformation = FALSE,
  Methods = c("BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson"),
  AnomalyDetection = NULL,
  XREGS = NULL,
  Lags = c(1L:5L),
  MA_Periods = c(2L:5L),
  SD_Periods = NULL,
  Skew_Periods = NULL,
  Kurt_Periods = NULL,
  Quantile_Periods = NULL,
  Quantiles_Selected = c("q5", "q95"),
  Difference = TRUE,
  FourierTerms = 6L,
  CalendarVariables = c("second", "minute", "hour", "wday", "mday", "yday", "week",
    "isoweek", "month", "quarter", "year"),
  HolidayVariable = c("USPublicHolidays", "EasterGroup", "ChristmasGroup",
    "OtherEcclesticalFeasts"),
  HolidayLags = 1L,
  HolidayMovingAverages = 1L:2L,
  TimeTrendVariable = FALSE,
  ZeroPadSeries = NULL,
  DataTruncate = FALSE,
```

```
  SplitRatios = c(0.7, 0.2, 0.1),
  TaskType = "GPU",
  NumGPU = 1,
  EvalMetric = "RMSE",
  EvalMetricValue = 1.5,
  LossFunction = "RMSE",
  LossFunctionValue = 1.5,
  GridTune = FALSE,
  PassInGrid = NULL,
  ModelCount = 100,
  MaxRunsWithoutNewWinner = 50,
  MaxRunMinutes = 24L * 60L,
  Langevin = FALSE,
  DiffusionTemperature = 10000,
  NTrees = 1000,
  L2_Leaf_Reg = 3,
  RandomStrength = 1,
  BorderCount = 254,
  Depth = 6,
  BootStrapType = c("Bayesian", "Bernoulli", "Poisson", "MVS", "No"),
  PartitionType = "timeseries",
  Timer = TRUE,
  DebugMode = FALSE
)
```

Arguments

`data`	Supply your full series data set here
`TimeWeights`	Supply a value that will be multiplied by he time trend value
`NonNegativePred`	TRUE or FALSE
`RoundPreds`	Rounding predictions to an integer value. TRUE or FALSE. Defaults to FALSE
`TrainOnFull`	Set to TRUE to train on full data
`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"
`HierarchGroups`	Vector of hierachy categorical columns.
`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. "1min", "5min", "10min", "15min", "30min", "hour", "day", "week", "month", "quarter", "year".
`TimeGroups`	Select time aggregations for adding various time aggregated GDL features.
`PDFOutputPath`	NULL or a path file to output PDFs to a specified folder
`SaveDataPath`	NULL Or supply a path. Data saved will be called 'ModelID'_data.csv
`NumOfParDepPlots`	Supply a number for the number of partial dependence plots you want returned

TargetTransformation
: Run AutoTransformationCreate() to find best transformation for the target variable. Tests YeoJohnson, BoxCox, and Asigh (also Asin and Logit for proportion target variables).

Methods
: Transformation options to test which include "BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson"

AnomalyDetection
: NULL for not using the service. Other, provide a list, e.g. AnomalyDetection = list("tstat_high" = 4, tstat_low = -4)

XREGS
: Additional data to use for model development and forecasting. Data needs to be a complete series which means both the historical and forward looking values over the specified forecast window needs to be supplied.

Lags
: Select the periods for all lag variables you want to create. E.g. c(1:5,52)

MA_Periods
: Select the periods for all moving average variables you want to create. E.g. c(1:5,52)

SD_Periods
: Select the periods for all moving standard deviation variables you want to create. E.g. c(1:5,52)

Skew_Periods
: Select the periods for all moving skewness variables you want to create. E.g. c(1:5,52)

Kurt_Periods
: Select the periods for all moving kurtosis variables you want to create. E.g. c(1:5,52)

Quantile_Periods
: Select the periods for all moving quantiles variables you want to create. E.g. c(1:5,52)

Quantiles_Selected
: Select from the following "q5", "q10", "q15", "q20", "q25", "q30", "q35", "q40", "q45", "q50", "q55", "q60", "q65", "q70", "q75", "q80", "q85", "q90", "q95"

Difference
: Puts the I in ARIMA for single series and grouped series.

FourierTerms
: Set to the max number of pairs. E.g. 2 means to generate two pairs for by each group level and interations if hierarchy is enabled.

CalendarVariables
: NULL, or select from "second", "minute", "hour", "wday", "mday", "yday", "week", "isoweek", "month", "quarter", "year"

HolidayVariable
: NULL, or select from "USPublicHolidays", "EasterGroup", "ChristmasGroup", "OtherEcclesticalFeasts"

HolidayLags
: Number of lags to build off of the holiday count variable.

HolidayMovingAverages
: Number of moving averages to build off of the holiday count variable.

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.

ZeroPadSeries
: Set to "all", "inner", or NULL. See TimeSeriesFill for explanation

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"
- `NumGPU` Defaults to 1. If CPU is set this argument will be ignored.
- `EvalMetric` Select from "RMSE", "MAE", "MAPE", "Poisson", "Quantile", "LogLinQuantile", "Lq", "NumErrors", "SMAPE", "R2", "MSLE", "MedianAbsoluteError"
- `EvalMetricValue` Used when EvalMetric accepts an argument. See `AutoCatBoostRegression`
- `LossFunction` Used in model training for model fitting. Select from 'RMSE', 'MAE', 'Quantile', 'LogLinQuantile', 'MAPE', 'Poisson', 'PairLogitPairwise', 'Tweedie', 'QueryRMSE'
- `LossFunctionValue` Used when LossFunction accepts an argument. See `AutoCatBoostRegression`
- `GridTune` Set to TRUE to run a grid tune
- `PassInGrid` Defaults to NULL
- `ModelCount` Set the number of models to try in the grid tune
- `MaxRunsWithoutNewWinner` Default is 50
- `MaxRunMinutes` Default is 60*60
- `Langevin` Enables the Stochastic Gradient Langevin Boosting mode. If TRUE and TaskType == "GPU" then TaskType will be converted to "CPU"
- `DiffusionTemperature` Default is 10000
- `NTrees` Select the number of trees you want to have built to train the model
- `L2_Leaf_Reg` l2 reg parameter
- `RandomStrength` Default is 1
- `BorderCount` Default is 254
- `Depth` Depth of catboost model
- `BootStrapType` Select from Catboost list
- `PartitionType` Select "random" for random data partitioning "timeseries" for partitioning by time frames
- `Timer` Set to FALSE to turn off the updating print statements for progress
- `DebugMode` Defaults to FALSE. Set to TRUE to get a print statement of each high level comment in function

Value

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

Author(s)

Adrian Antico

See Also

Other Automated Panel Data Forecasting: `AutoCatBoostHurdleCARMA()`, `AutoCatBoostVectorCARMA()`, `AutoH2OCARMA()`, `AutoXGBoostCARMA()`

Examples

```
## Not run:

# Set up path
Path <- "C:/Users/Bizon/Documents/GitHub"

# Set up environment
data.table::setDTthreads(percent = 100)

# Load data
data <- data.table::fread(file = file.path(Path, "walmart.csv"),index = c("Store","Dept"))

# Set negative numbers to 0
data <- data[, Weekly_Sales := data.table::fifelse(Weekly_Sales < 0, 0, Weekly_Sales)]

# Subset for Stores / Departments with Full Series Available: (143 time points each)----
data <- data[, Counts := .N, by = c("Store","Dept")][Counts == 143][, Counts := NULL]

# Subset Columns (remove IsHoliday column)----
data <- data[, .SD, .SDcols = c("Store","Dept","Date","Weekly_Sales")]

# Setup xregs
xregs <- data[, .SD, .SDcols = c("Date","Store","Dept")]

# Change data types
data[, ":=" (Store = as.character(Store), Dept = as.character(Dept))]
xregs[, ":=" (Store = as.character(Store), Dept = as.character(Dept))]

# Add GroupVar to xregs
xregs[, GroupVar := do.call(paste, c(.SD, sep = " ")), .SDcols = c("Store","Dept")]

# Change names of categoricals in xregs
data.table::setnames(xregs, c("Store","Dept"), c("STORE","DEPT"))

# Subset data so we have an out of time sample
data1 <- data.table::copy(data[, ID := 1:.N, by = c("Store","Dept")][ID <= 125][, ID := NULL])
data[, ID := NULL]

# Define Holdout windows
N <- data1[, .N, by = c("Store","Dept")][1, N]
N1 <- xregs[, .N, by = c("STORE","DEPT")][1, N]

# Setup Grid Tuning & Feature Tuning
Tuning <- data.table::CJ(
  TimeWeights = c("None",0.9999,0.999,0.99),
  HierachGroups = c("TRUE","FALSE"),
  MaxTimeGroups = c("weeks","months","quarters"),
  TargetTransformation = c("TRUE","FALSE"),
  Difference = c("TRUE","FALSE"),
  TimeTrendVariable = c("TRUE","FALSE"),
  EvalMetric = c("RMSE","Huber"),
  LossFunction = c("RMSE","Huber"),
  Langevin = c("TRUE","FALSE"),
  L2_Leaf_Reg = c(1.0,2.0,3.0,4.0))

# Plot list
```

```
PlotList <- list()

# Total runs
TotalRuns <- Tuning[,.N]

# Run models
for(Run in seq_len(TotalRuns)) {

  # Print Run
  for(zz in seq_len(100)) print(Run)

  # Use clean data each run
  xregs_new <- data.table::copy(xregs)
  data_new <- data.table::copy(data1)

  # Timer
  StartTime <- Sys.time()

  # Run carma system
  Results <- RemixAutoML::AutoCatBoostCARMA(

    # data args
    data = data_new,
    TimeWeights = if(Tuning[Run, TimeWeights] == "None") NULL else as.numeric(Tuning[Run, TimeWeights]),
    TargetColumnName = "Weekly_Sales",
    DateColumnName = "Date",
    HierarchGroups = if(as.logical(Tuning[Run, HierachGroups])) c("Store","Dept") else NULL,
    GroupVariables = c("Store","Dept"),
    TimeUnit = "weeks",
    TimeGroups = if(Tuning[Run, MaxTimeGroups] == "weeks") "weeks" else if(Tuning[Run, MaxTimeGroups] == "months

    # Production args
    TrainOnFull = TRUE,
    SplitRatios = c(N / N1, 1 - N / N1),
    PartitionType = "random",
    FC_Periods = N1-N,
    TaskType = "GPU",
    NumGPU = 1,
    Timer = TRUE,
    DebugMode = TRUE,

    # Target transformations
    TargetTransformation = as.logical(Tuning[Run, TargetTransformation]),
    Methods = c("BoxCox","Asinh","Log","LogPlus1","YeoJohnson"),
    Difference = as.logical(Tuning[Run, Difference]),
    NonNegativePred = TRUE,
    RoundPreds = as.logical(Tuning[Run, RoundPreds]),

    # Calendar features
    CalendarVariables = c("week","wom","month","quarter"),
    HolidayVariable = c("USPublicHolidays","EasterGroup","ChristmasGroup","OtherEcclesticalFeasts"),
    HolidayLags = c(1,2,3),
    HolidayMovingAverages = c(2,3),

    # Time series features
    Lags = list("weeks" = c(1,2,3,4,5,8,9,12,13,51,52,53), "months" = c(1,2,6,12)),
    MA_Periods = list("weeks" = c(2,3,4,5,8,9,12,13,51,52,53), "months" = c(2,6,12)),
```

```
  SD_Periods = NULL,
  Skew_Periods = NULL,
  Kurt_Periods = NULL,
  Quantile_Periods = NULL,
  Quantiles_Selected = NULL,

  # Bonus features
  AnomalyDetection = NULL,
  XREGS = xregs_new,
  FourierTerms = 0,
  TimeTrendVariable = as.logical(Tuning[Run, TimeTrendVariable]),
  ZeroPadSeries = NULL,
  DataTruncate = FALSE,

  # ML evaluation output
  PDFOutputPath = NULL,
  SaveDataPath = NULL,
  NumOfParDepPlots = 0L,

  # ML loss functions
  EvalMetric = Tuning[Run, EvalMetric],
  EvalMetricValue = 10,
  LossFunction = Tuning[Run, LossFunction],
  LossFunctionValue = 10,

  # ML grid tuning args
  GridTune = FALSE,
  PassInGrid = NULL,
  ModelCount = 5,
  MaxRunsWithoutNewWinner = 50,
  MaxRunMinutes = 60*60,

  # ML tuning args
  NTrees = 12000,
  Depth = 9,
  L2_Leaf_Reg = Tuning[Run, L2_Leaf_Reg],
  Langevin = as.logical(Tuning[Run, Langevin]),
  DiffusionTemperature = 10000,
  RandomStrength = 1,
  BorderCount = 254,
  BootStrapType = c("Bayesian","Bernoulli","Poisson","MVS","No"))

# Timer
EndTime <- Sys.time()

# Prepare data for evaluation
Results <- Results$Forecast
data.table::setnames(Results, "Weekly_Sales", "Old")
Results <- merge(Results, data, by = c("Store","Dept","Date"), all = FALSE)
Results <- Results[is.na(Old)]
Results[, Old := NULL]

# Create totals and subtotals
Results <- data.table::groupingsets(
  x = Results,
  j = list(Predictions = sum(Predictions), Weekly_Sales = sum(Weekly_Sales)),
  by = c("Date", "Store", "Dept"),
```

```
    sets = list(c("Date", "Store", "Dept"), c("Store", "Dept"), "Store", "Dept", "Date"))
  Results[, Store := data.table::fifelse(is.na(Store), "Total", Store)]
  Results[, Dept := data.table::fifelse(is.na(Dept), "Total", Dept)]

  # Add error measures
  Results[, Weekly_MAE := abs(Weekly_Sales - Predictions)]
  Results[, Weekly_MAPE := Weekly_MAE / Weekly_Sales]

  # Weekly results
  Weekly_MAPE <- Results[, list(Weekly_MAPE = mean(Weekly_MAPE)), by = list(Store,Dept)]

  # Monthly results
  temp <- data.table::copy(Results)
  temp <- temp[, Date := lubridate::floor_date(Date, unit = "months")]
  temp <- temp[, lapply(.SD, sum), by = c("Date","Store","Dept"), .SDcols = c("Predictions", "Weekly_Sales")]
  temp[, Monthly_MAE := abs(Weekly_Sales - Predictions)]
  temp[, Monthly_MAPE := Monthly_MAE / Weekly_Sales]
  Monthly_MAPE <- temp[, list(Monthly_MAPE = mean(Monthly_MAPE)), by = list(Store,Dept)]

  # Create ts plot of actuals and predicted
  Totals <- Results[Store == "Total" & Dept == "Total"]
  Totals <- data.table::melt.data.table(data = Totals, id.vars = "Date", measure.vars = c("Predictions","Weekly
  PlotList[[Run]] <- eval(ggplot2::ggplot(data = Totals, ggplot2::aes(x = Date, y = Weekly_Sales, color = Series
                            ggplot2::geom_line() +
                            ggplot2::scale_color_manual(values = c("red","blue")) +
                            ggplot2::labs(
                              title = "Walmart Data Forecast",
                    subtitle = paste0("Weekly MAPE = ", round(100 * Weekly_MAPE[Store == "Total" & Dept == "Tot
                            RemixAutoML::ChartTheme(Size = 10, AngleX = 0, AngleY = 0))

  # Collect metrics
  Metrics <- data.table::data.table(
    RunNumber = Run,
    Total_Weekly_MAPE = Weekly_MAPE[Store == "Total" & Dept == "Total", Weekly_MAPE],
    Total_Monthly_MAPE = Monthly_MAPE[Store == "Total" & Dept == "Total", Monthly_MAPE],
    Tuning[Run],
    RunTime = EndTime - StartTime)

  # Append to file
  data.table::fwrite(Metrics, file = file.path(Path, "Walmart_CARMA_Metrics.csv"), append = TRUE)
}

## End(Not run)
```

`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,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  PrimaryDateColumn = NULL,
  ClassWeights = c(1, 1),
  IDcols = NULL,
  task_type = "GPU",
  NumGPUs = 1,
  eval_metric = "MCC",
  loss_function = NULL,
  model_path = NULL,
  metadata_path = NULL,
  SaveInfoToPDF = FALSE,
  ModelID = "FirstModel",
  NumOfParDepPlots = 0L,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  PassInGrid = NULL,
  GridTune = FALSE,
  MaxModelsInGrid = 10L,
  MaxRunsWithoutNewWinner = 20L,
  MaxRunMinutes = 24L * 60L,
  Shuffles = 1L,
  BaselineComparison = "default",
  MetricPeriods = 10L,
  langevin = FALSE,
  diffusion_temperature = 10000,
  Trees = 50L,
  Depth = 6,
  LearningRate = NULL,
  L2_Leaf_Reg = 3,
  RandomStrength = 1,
  BorderCount = 128,
  RSM = NULL,
  BootStrapType = NULL,
  GrowPolicy = NULL
)
```

Arguments

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

`TrainOnFull` Set to TRUE to train on full data and skip over evaluation steps

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

`NumGPUs` Numeric. If you have 4 GPUs supply 4 as a value.

`eval_metric` This is the metric used inside catboost to measure performance on validation data during a grid-tune. "AUC" is the default. 'Logloss', 'CrossEntropy', 'Precision', 'Recall', 'F1', 'BalancedAccuracy', 'BalancedErrorRate', 'MCC', 'Accuracy', 'CtrFactor', 'AUC', 'BrierScore', 'HingeLoss', 'HammingLoss', 'ZeroOneLoss', 'Kappa', 'WKappa', 'LogLikelihoodOfPrediction', 'TotalF1', 'PairLogit', 'PairLogitPairwise', 'PairAccuracy', 'QueryCrossEntropy', 'QuerySoftMax', 'PFound', 'NDCG', 'AverageGain', 'PrecisionAt', 'RecallAt', 'MAP'

`loss_function` Default is NULL. Select the loss function of choice. c("MultiRMSE", 'Logloss','CrossEntropy','Lq','

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

`metadata_path` A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to model_path.

`SaveInfoToPDF` Set to TRUE to save modeling information to PDF. If model_path or metadata_path aren't defined then output will be saved to the working directory

`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)

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.

MaxModelsInGrid — Number of models to test from grid options.

MaxRunsWithoutNewWinner — A number

MaxRunMinutes — In minutes

Shuffles — Numeric. List a number to let the program know how many times you want to shuffle the grids for grid tuning

BaselineComparison — Set to either "default" or "best". Default is to compare each successive model build to the baseline model using max trees (from function args). Best makes the comparison to the current best model.

MetricPeriods — Number of trees to build before evaluating intermediate metrics. Default is 10L

langevin — TRUE or FALSE. TRUE enables

diffusion_temperature — Default value is 10000

Trees — Bandit grid partitioned. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the trees numbers you want to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(1000L, 10000L, 1000L)

Depth — Bandit grid partitioned Number, or vector for depth to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(4L, 16L, 2L)

LearningRate — Bandit grid partitioned. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the LearningRate values to test. For running grid tuning, a NULL value supplied will mean these values are tested c(0.01,0.02,0.03,0.04)

L2_Leaf_Reg — Random testing. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the L2_Leaf_Reg values to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(1.0, 10.0, 1.0)

RandomStrength — A multiplier of randomness added to split evaluations. Default value is 1 which adds no randomness.

BorderCount — Number of splits for numerical features. Catboost defaults to 254 for CPU and 128 for GPU

RSM — CPU only. Random testing. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the RSM values to test. For running grid tuning, a NULL value supplied will mean these values are tested c(0.80, 0.85, 0.90, 0.95, 1.0)

BootStrapType — Random testing. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the BootStrapType values to test. For running grid tuning, a NULL value supplied will mean these values are tested c("Bayesian", "Bernoulli", "Poisson", "MVS", "No")

GrowPolicy — Random testing. NULL, character, or vector for GrowPolicy to test. For grid tuning, supply a vector of values. For running grid tuning, a NULL value supplied will mean these values are tested c("SymmetricTree", "Depthwise", "Lossguide")

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 Automated Supervised Learning - Binary Classification: AutoH2oDRFClassifier(), AutoH2oGAMClassifier(), AutoH2oGBMClassifier(), AutoH2oGLMClassifier(), AutoH2oMLClassifier(), AutoXGBoostClassifier()

Examples

```
## Not run:
# Create some dummy correlated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 10000,
  ID = 2,
  ZIP = 0,
  AddDate = FALSE,
  Classification = TRUE,
  MultiClass = FALSE)

# Run function
TestModel <- RemixAutoML::AutoCatBoostClassifier(

    # GPU or CPU and the number of available GPUs
    task_type = "GPU",
    NumGPUs = 1,

    # Metadata arguments:
    #   'ModelID' is used to create part of the file
    #       names generated when saving to file'
    #   'model_path' is where the minimal model objects
    #       for scoring will be stored
    #   'ModelID' will be the name of the saved model object
    #   'metadata_path' is where model evaluation and model
    #       interpretation files are saved
    #    objects saved to model_path if metadata_path is null
    #    Saved objects include:
    #    'ModelID_ValidationData.csv' is the supplied or generated
    #       TestData with predicted values
    #    'ModelID_ROC_Plot.png' and 'Model_ID_EvaluationPlot.png'
    #        calibration plot
    #    'ModelID_VariableImportance.csv' is the variable importance.
    #        This won't be saved to file if GrowPolicy is either
    #          "Depthwise" or "Lossguide" was used
    #    'ModelID_ExperimentGrid.csv' if GridTune = TRUE.
    #        Results of all model builds including parameter settings,
    #          bandit probs, and grid IDs
    #    'ModelID_EvaluationMetrics.csv' which contains all confusion
    #           matrix measures across all thresholds
```

```
ModelID = "Test_Model_1",
model_path = normalizePath("./"),
metadata_path = file.path(normalizePath("./")),
SaveModelObjects = FALSE,
ReturnModelObjects = TRUE,
SaveInfoToPDF = FALSE,

# Data arguments:
#   'TrainOnFull' is to train a model with 100 percent of
#      your data.
#   That means no holdout data will be used for evaluation
#   If ValidationData and TestData are NULL and TrainOnFull
#      is FALSE then data will be split 70 20 10
#   'PrimaryDateColumn' is a date column in data that is
#      meaningful when sorted.
#    CatBoost categorical treatment is enhanced when supplied
#   'IDcols' are columns in your data that you don't use for
#      modeling but get returned with ValidationData
data = data,
TrainOnFull = FALSE,
ValidationData = NULL,
TestData = NULL,
TargetColumnName = "Adrian",
FeatureColNames = names(data)[!names(data) %chin%
    c("IDcol_1","IDcol_2","Adrian")],
PrimaryDateColumn = NULL,
ClassWeights = c(1L,1L),
IDcols = c("IDcol_1","IDcol_2"),

# Model evaluation:
#   'eval_metric' is the measure catboost uses when evaluting
#       on holdout data during its bandit style process
#   'loss_function' the loss function used in training optimization
#   'NumOfParDepPlots' Number of partial dependence calibration plots
#       generated.
#     A value of 3 will return plots for the top 3 variables based
#       on variable importance
#     Won't be returned if GrowPolicy is either "Depthwise" or
#       "Lossguide" is used
#     Can run the RemixAutoML::ParDepCalPlots() with the outputted
#        ValidationData
eval_metric = "AUC",
loss_function = "Logloss",
MetricPeriods = 10L,
NumOfParDepPlots = ncol(data)-1L-2L,

# Grid tuning arguments:
#   'PassInGrid' is for retraining using a previous grid winning args
#   'MaxModelsInGrid' is a cap on the number of models that will run
#   'MaxRunsWithoutNewWinner' number of runs without a new winner
#      before exiting grid tuning
#   'MaxRunMinutes' is a cap on the number of minutes that will run
#   'Shuffles' is the number of times you want the random grid
#      arguments shuffled
#   'BaselineComparison' default means to compare each model build
#      with a default built of catboost using max(Trees)
#   'MetricPeriods' is the number of trees built before evaluting
```

```
    #      holdoutdata internally. Used in finding actual Trees used.
    PassInGrid = NULL,
    GridTune = FALSE,
    MaxModelsInGrid = 100L,
    MaxRunsWithoutNewWinner = 20L,
    MaxRunMinutes = 24L*60L,
    Shuffles = 4L,
    BaselineComparison = "default",

    # Trees, Depth, and LearningRate used in the bandit grid tuning
    # Must set Trees to a single value if you are not grid tuning
    # The ones below can be set to NULL and the values in the example
    #   will be used
    # GrowPolicy is turned off for CPU runs
    # BootStrapType utilizes Poisson only for GPU and MVS only for CPU
    langevin = FALSE,
    diffusion_temperature = 10000,
    Trees = seq(100L, 500L, 50L),
    Depth = seq(4L, 8L, 1L),
    LearningRate = seq(0.01,0.10,0.01),
    L2_Leaf_Reg = seq(1.0, 10.0, 1.0),
    RandomStrength = 1,
    BorderCount = 128,
    RSM = c(0.80, 0.85, 0.90, 0.95, 1.0),
    BootStrapType = c("Bayesian", "Bernoulli", "Poisson", "MVS", "No"),
    GrowPolicy = c("SymmetricTree", "Depthwise", "Lossguide"))

# Output
TestModel$Model
TestModel$ValidationData
TestModel$ROC_Plot
TestModel$EvaluationPlot
TestModel$EvaluationMetrics
TestModel$VariableImportance
TestModel$InteractionImportance
TestModel$ShapValuesDT
TestModel$VI_Plot
TestModel$PartialDependencePlots
TestModel$GridMetrics
TestModel$ColNames

## End(Not run)
```

`AutoCatBoostFreqSizeScoring` *AutoCatBoostFreqSizeScoring is for scoring the models build with AutoCatBoostSizeFreqDist()*

Description

AutoCatBoostFreqSizeScoring is for scoring the models build with AutoCatBoostSizeFreqDist(). It will return the predicted values for every quantile model for both distributions for 1 to the max forecast periods you provided to build the scoring data.

Usage

```
AutoCatBoostFreqSizeScoring(
  ScoringData,
  TargetColumnNames = NULL,
  FeatureColumnNames = NULL,
  IDcols = NULL,
  CountQuantiles = seq(0.1, 0.9, 0.1),
  SizeQuantiles = seq(0.1, 0.9, 0.1),
  ModelPath = NULL,
  ModelIDs = c("CountModel", "SizeModel"),
  KeepFeatures = TRUE
)
```

Arguments

`ScoringData` The scoring data returned from IntermittentDemandScoringDataGenerator()

`TargetColumnNames` A character or numeric vector of the target names. E.g. c("Counts","TARGET_qty")

`FeatureColumnNames` A character vector of column names or column numbers

`IDcols` ID columns you want returned with the data that is not a model feature

`CountQuantiles` A numerical vector of the quantiles used in model building

`SizeQuantiles` A numerical vector of the quantiles used in model building

`ModelPath` The path file to where you models were saved

`ModelIDs` The ID's used in model building

`KeepFeatures` Set to TRUE to return the features with the predicted values

Value

Returns a list of CountData scores, SizeData scores, along with count and size prediction column names

Author(s)

Adrian Antico

See Also

Other Automated Time Series: `AutoBanditNNet()`, `AutoBanditSarima()`, `AutoH2oGBMFreqSizeScoring()`, `AutoTBATS()`, `AutoTS()`

Examples

```
## Not run:
FinalData <- AutoCatBoostFreqSizeScoring(
  ScoringData,
  TargetColumnNames = c("Counts","TARGET_qty"),
  FeatureColumnNames = 1:ncol(ScoringData),
  IDcols = NULL,
  CountQuantiles = seq(0.10,0.90,0.10),
  SizeQuantiles = seq(0.10,0.90,0.10),
```

```
  ModelPath = getwd(),
  ModelIDs = c("CountModel","SizeModel"),
  KeepFeatures = TRUE)

## End(Not run)
```

AutoCatBoostHurdleCARMA

AutoCatBoostHurdleCARMA

Description

AutoCatBoostHurdleCARMA is an intermittent demand, Mutlivariate Forecasting algorithms with calendar variables, Holiday counts, holiday lags, holiday moving averages, differencing, transformations, interaction-based categorical encoding using target variable and features to generate various time-based aggregated lags, moving averages, moving standard deviations, moving skewness, moving kurtosis, moving quantiles, parallelized interaction-based fourier pairs by grouping variables, and Trend Variables.

Usage

```
AutoCatBoostHurdleCARMA(
  data,
  NonNegativePred = FALSE,
  Threshold = NULL,
  RoundPreds = FALSE,
  TrainOnFull = FALSE,
  TargetColumnName = "Target",
  DateColumnName = "DateTime",
  HierarchGroups = NULL,
  GroupVariables = NULL,
  FC_Periods = 30,
  TimeUnit = "week",
  TimeGroups = c("weeks", "months"),
  NumOfParDepPlots = 10L,
  TargetTransformation = FALSE,
 Methods = c("BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson"),
  AnomalyDetection = NULL,
  XREGS = NULL,
  Lags = c(1L:5L),
  MA_Periods = c(2L:5L),
  SD_Periods = NULL,
  Skew_Periods = NULL,
  Kurt_Periods = NULL,
  Quantile_Periods = NULL,
  Quantiles_Selected = c("q5", "q95"),
  Difference = TRUE,
  FourierTerms = 6L,
 CalendarVariables = c("second", "minute", "hour", "wday", "mday", "yday", "week",
    "isoweek", "month", "quarter", "year"),
  HolidayVariable = c("USPublicHolidays", "EasterGroup", "ChristmasGroup",
```

```
    "OtherEcclesticalFeasts"),
  HolidayLags = 1L,
  HolidayMovingAverages = 1L:2L,
  TimeTrendVariable = FALSE,
  ZeroPadSeries = NULL,
  DataTruncate = FALSE,
  SplitRatios = c(0.7, 0.2, 0.1),
  TaskType = "GPU",
  NumGPU = 1,
  EvalMetric = "RMSE",
  GridTune = FALSE,
  PassInGrid = NULL,
  ModelCount = 100,
  MaxRunsWithoutNewWinner = 50,
  MaxRunMinutes = 24L * 60L,
  NTrees = list(classifier = seq(1000, 2000, 100), regression = seq(1000, 2000, 100)),
  Depth = list(classifier = seq(6, 10, 1), regression = seq(6, 10, 1)),
  LearningRate = list(classifier = seq(0.01, 0.25, 0.01), regression = seq(0.01, 0.25,
    0.01)),
  L2_Leaf_Reg = list(classifier = 3:6, regression = 3:6),
  RandomStrength = list(classifier = 1:10, regression = 1:10),
  BorderCount = list(classifier = seq(32, 256, 16), regression = seq(32, 256, 16)),
  BootStrapType = c("Bayesian", "Bernoulli", "Poisson", "MVS", "No"),
  PartitionType = "timeseries",
  Timer = TRUE,
  DebugMode = FALSE
)
```

Arguments

`data`	Supply your full series data set here
`NonNegativePred`	TRUE or FALSE
`Threshold`	Select confusion matrix measure to optimize for pulling in threshold. Choose from "MCC", "Acc", "TPR", "TNR", "FNR", "FPR", "FDR", "FOR", "F1_Score", "F2_Score", "F0.5_Score", "NPV", "PPV", "ThreatScore", "Utility"
`RoundPreds`	Rounding predictions to an integer value. TRUE or FALSE. Defaults to FALSE
`TrainOnFull`	Set to TRUE to train on full data
`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"
`HierarchGroups`	Vector of hierachy categorical columns.
`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. "1min", "5min", "10min", "15min", "30min", "hour", "day", "week", "month", "quarter", "year".
`TimeGroups`	Select time aggregations for adding various time aggregated GDL features.

NumOfParDepPlots
: Supply a number for the number of partial dependence plots you want returned

TargetTransformation
: Run AutoTransformationCreate() to find best transformation for the target variable. Tests YeoJohnson, BoxCox, and Asigh (also Asin and Logit for proportion target variables).

Methods
: Transformation options to test which include "BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson"

AnomalyDetection
: NULL for not using the service. Other, provide a list, e.g. AnomalyDetection = list("tstat_high" = 4, tstat_low = -4)

XREGS
: Additional data to use for model development and forecasting. Data needs to be a complete series which means both the historical and forward looking values over the specified forecast window needs to be supplied.

Lags
: Select the periods for all lag variables you want to create. E.g. c(1:5,52)

MA_Periods
: Select the periods for all moving average variables you want to create. E.g. c(1:5,52)

SD_Periods
: Select the periods for all moving standard deviation variables you want to create. E.g. c(1:5,52)

Skew_Periods
: Select the periods for all moving skewness variables you want to create. E.g. c(1:5,52)

Kurt_Periods
: Select the periods for all moving kurtosis variables you want to create. E.g. c(1:5,52)

Quantile_Periods
: Select the periods for all moving quantiles variables you want to create. E.g. c(1:5,52)

Quantiles_Selected
: Select from the following "q5", "q10", "q15", "q20", "q25", "q30", "q35", "q40", "q45", "q50", "q55", "q60", "q65", "q70", "q75", "q80", "q85", "q90", "q95"

Difference
: Puts the I in ARIMA for single series and grouped series.

FourierTerms
: Set to the max number of pairs. E.g. 2 means to generate two pairs for by each group level and interations if hierarchy is enabled.

CalendarVariables
: NULL, or select from "second", "minute", "hour", "wday", "mday", "yday", "week", "isoweek", "month", "quarter", "year"

HolidayVariable
: NULL, or select from "USPublicHolidays", "EasterGroup", "ChristmasGroup", "OtherEcclesticalFeasts"

HolidayLags
: Number of lags to build off of the holiday count variable.

HolidayMovingAverages
: Number of moving averages to build off of the holiday count variable.

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.

ZeroPadSeries
: Set to "all", "inner", or NULL. See TimeSeriesFill for explanation

`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"
`NumGPU`	Defaults to 1. If CPU is set this argument will be ignored.
`EvalMetric`	Select from "RMSE", "MAE", "MAPE", "Poisson", "Quantile", "LogLinQuantile", "Lq", "NumErrors", "SMAPE", "R2", "MSLE", "MedianAbsoluteError"
`GridTune`	Set to TRUE to run a grid tune
`PassInGrid`	Defaults to NULL
`ModelCount`	Set the number of models to try in the grid tune
`MaxRunsWithoutNewWinner`	Default is 50
`MaxRunMinutes`	Default is 60*60
`NTrees`	Select the number of trees you want to have built to train the model
`Depth`	Depth of catboost model
`LearningRate`	learning_rate
`L2_Leaf_Reg`	l2 reg parameter
`RandomStrength`	Default is 1
`BorderCount`	Default is 254
`BootStrapType`	Select from Catboost list
`PartitionType`	Select "random" for random data partitioning "timeseries" for partitioning by time frames
`Timer`	Set to FALSE to turn off the updating print statements for progress
`DebugMode`	Defaults to FALSE. Set to TRUE to get a print statement of each high level comment in function

Value

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

Author(s)

Adrian Antico

See Also

Other Automated Panel Data Forecasting: `AutoCatBoostCARMA()`, `AutoCatBoostVectorCARMA()`, `AutoH2OCARMA()`, `AutoXGBoostCARMA()`

Examples

```
## Not run:

 # Single group variable and xregs ----

 # Load Walmart Data from Dropbox----
 data <- data.table::fread(
   "https://www.dropbox.com/s/2str3ek4f4cheqi/walmart_train.csv?dl=1")

 # Subset for Stores / Departments With Full Series
 data <- data[, Counts := .N, by = c("Store","Dept")][Counts == 143][
   , Counts := NULL]

 # Subset Columns (remove IsHoliday column)----
 keep <- c("Store","Dept","Date","Weekly_Sales")
 data <- data[, ..keep]
 data <- data[Store == 1][, Store := NULL]
 xregs <- data.table::copy(data)
 data.table::setnames(xregs, "Dept", "GroupVar")
 data.table::setnames(xregs, "Weekly_Sales", "Other")
 data <- data[as.Date(Date) < as.Date('2012-09-28')]

 # Add zeros for testing
 data[runif(.N) < 0.25, Weekly_Sales := 0]

 # Build forecast
 CatBoostResults <- RemixAutoML::AutoCatBoostHurdleCARMA(

  # data args
  data = data, # TwoGroup_Data,
  TargetColumnName = "Weekly_Sales",
  DateColumnName = "Date",
  HierarchGroups = NULL,
  GroupVariables = c("Dept"),
  TimeUnit = "weeks",
  TimeGroups = c("weeks","months"),

  # Production args
  TrainOnFull = FALSE,
  SplitRatios = c(1 - 20 / 138, 10 / 138, 10 / 138),
  PartitionType = "random",
  FC_Periods = 4,
  Timer = TRUE,
  DebugMode = TRUE,

  # Target transformations
  TargetTransformation = TRUE,
  Methods = c("BoxCox", "Asinh", "Asin", "Log",
    "LogPlus1", "Logit", "YeoJohnson"),
  Difference = FALSE,
  NonNegativePred = FALSE,
  RoundPreds = FALSE,

  # Date features
  CalendarVariables = c("week", "month", "quarter"),
  HolidayVariable = c("USPublicHolidays",
```

```
    "EasterGroup",
    "ChristmasGroup","OtherEcclesticalFeasts"),
  HolidayLags = 1,
  HolidayMovingAverages = 1:2,

  # Time series features
  Lags = list("weeks" = seq(2L, 10L, 2L),
    "months" = c(1:3)),
  MA_Periods = list("weeks" = seq(2L, 10L, 2L),
    "months" = c(2,3)),
  SD_Periods = NULL,
  Skew_Periods = NULL,
  Kurt_Periods = NULL,
  Quantile_Periods = NULL,
  Quantiles_Selected = c("q5","q95"),

  # Bonus features
  AnomalyDetection = NULL,
  XREGS = xregs,
  FourierTerms = 2,
  TimeTrendVariable = TRUE,
  ZeroPadSeries = NULL,
  DataTruncate = FALSE,

  # ML Args
  NumOfParDepPlots = 100L,
  EvalMetric = "RMSE",
  GridTune = FALSE,
  PassInGrid = NULL,
  ModelCount = 5,
  TaskType = "GPU",
  NumGPU = 1,
  MaxRunsWithoutNewWinner = 50,
  MaxRunMinutes = 60*60,
  NTrees = 2500,
  L2_Leaf_Reg = 3.0,
  LearningRate = list("classifier" = seq(0.01,0.25,0.01), "regression" = seq(0.01,0.25,0.01)),
  RandomStrength = 1,
  BorderCount = 254,
  BootStrapType = c("Bayesian", "Bernoulli", "Poisson", "MVS", "No"),
  Depth = 6)

# Two group variables and xregs

# Load Walmart Data from Dropbox----
data <- data.table::fread(
 "https://www.dropbox.com/s/2str3ek4f4cheqi/walmart_train.csv?dl=1")

# Subset for Stores / Departments With Full Series
data <- data[, Counts := .N, by = c("Store","Dept")][Counts == 143][
  , Counts := NULL]

# Put negative values at 0
data[, Weekly_Sales := data.table::fifelse(Weekly_Sales < 0, 0, Weekly_Sales)]

# Subset Columns (remove IsHoliday column)----
keep <- c("Store","Dept","Date","Weekly_Sales")
```

```
data <- data[, ..keep]
data <- data[Store %in% c(1,2)]

xregs <- data.table::copy(data)
xregs[, GroupVar := do.call(paste, c(.SD, sep = " ")), .SDcols = c("Store","Dept")]
xregs[, c("Store","Dept") := NULL]
data.table::setnames(xregs, "Weekly_Sales", "Other")
xregs[, Other := jitter(Other, factor = 25)]
data <- data[as.Date(Date) < as.Date('2012-09-28')]

# Add some zeros for testing
data[runif(.N) < 0.25, Weekly_Sales := 0]

# Build forecast
Output <- RemixAutoML::AutoCatBoostHurdleCARMA(

  # data args
  data = data,
  TargetColumnName = "Weekly_Sales",
  DateColumnName = "Date",
  HierarchGroups = NULL,
  GroupVariables = c("Store","Dept"),
  TimeUnit = "weeks",
  TimeGroups = c("weeks","months"),

  # Production args
  TrainOnFull = TRUE,
  SplitRatios = c(1 - 20 / 138, 10 / 138, 10 / 138),
  PartitionType = "random",
  FC_Periods = 4,
  Timer = TRUE,
  DebugMode = TRUE,

  # Target transformations
  TargetTransformation = TRUE,
  Methods = c("BoxCox", "Asinh", "Asin", "Log",
              "LogPlus1", "Logit", "YeoJohnson"),
  Difference = FALSE,
  NonNegativePred = FALSE,
  Threshold = NULL,
  RoundPreds = FALSE,

  # Date features
  CalendarVariables = c("week", "month", "quarter"),
  HolidayVariable = c("USPublicHolidays",
                      "EasterGroup",
                      "ChristmasGroup","OtherEcclesticalFeasts"),
  HolidayLags = 1,
  HolidayMovingAverages = 1:2,

  # Time series features
  Lags = list("weeks" = seq(2L, 10L, 2L),
              "months" = c(1:3)),
  MA_Periods = list("weeks" = seq(2L, 10L, 2L),
                    "months" = c(2,3)),
  SD_Periods = NULL,
  Skew_Periods = NULL,
```

```
  Kurt_Periods = NULL,
  Quantile_Periods = NULL,
  Quantiles_Selected = c("q5","q95"),

  # Bonus features
  AnomalyDetection = NULL,
  XREGS = xregs,
  FourierTerms = 2,
  TimeTrendVariable = TRUE,
  ZeroPadSeries = NULL,
  DataTruncate = FALSE,

  # ML Args
  NumOfParDepPlots = 100L,
  EvalMetric = "RMSE",
  GridTune = FALSE,
  PassInGrid = NULL,
  ModelCount = 5,
  TaskType = "GPU",
  NumGPU = 1,
  MaxRunsWithoutNewWinner = 50,
  MaxRunMinutes = 60*60,
  NTrees = list("classifier" = seq(1000,2000,100), "regression" = seq(1000,2000,100)),
  Depth = list("classifier" = seq(6,10,1), "regression" = seq(6,10,1)),
  LearningRate = list("classifier" = seq(0.01,0.25,0.01), "regression" = seq(0.01,0.25,0.01)),
  L2_Leaf_Reg = list("classifier" = 3.0:6.0, "regression" = 3.0:6.0),
  RandomStrength = list("classifier" = 1:10, "regression" = 1:10),
  BorderCount = list("classifier" = seq(32,256,16), "regression" = seq(32,256,16)),
  BootStrapType = c("Bayesian", "Bernoulli", "Poisson", "MVS", "No"))

## End(Not run)
```

AutoCatBoostHurdleModel

AutoCatBoostHurdleModel for generalized hurdle modeling

Description

AutoCatBoostHurdleModel for generalized hurdle modeling. Check out the Readme.Rd on github for more background.

Usage

```
AutoCatBoostHurdleModel(
  data = NULL,
  TimeWeights = NULL,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  Buckets = 0L,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  PrimaryDateColumn = NULL,
```

```
  IDcols = NULL,
  TransformNumericColumns = NULL,
  Methods = c("BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson"),
  ClassWeights = NULL,
  SplitRatios = c(0.7, 0.2, 0.1),
  task_type = "GPU",
  ModelID = "ModelTest",
  Paths = NULL,
  MetaDataPaths = NULL,
  SaveModelObjects = FALSE,
  ReturnModelObjects = TRUE,
  NumOfParDepPlots = 10L,
  PassInGrid = NULL,
  GridTune = FALSE,
  BaselineComparison = "default",
  MaxModelsInGrid = 1L,
  MaxRunsWithoutNewWinner = 20L,
  MaxRunMinutes = 60L * 60L,
  Shuffles = 2L,
  MetricPeriods = 25L,
  Langevin = FALSE,
  DiffusionTemperature = 10000,
  Trees = list(classifier = seq(1000, 2000, 100), regression = seq(1000, 2000, 100)),
  Depth = list(classifier = seq(6, 10, 1), regression = seq(6, 10, 1)),
  RandomStrength = list(classifier = seq(1, 10, 1), regression = seq(1, 10, 1)),
  BorderCount = list(classifier = seq(32, 256, 16), regression = seq(32, 256, 16)),
  LearningRate = list(classifier = seq(0.01, 0.25, 0.01), regression = seq(0.01, 0.25,
    0.01)),
  L2_Leaf_Reg = list(classifier = seq(3, 10, 1), regression = seq(1, 10, 1)),
  RSM = list(classifier = c(0.8, 0.85, 0.9, 0.95, 1), regression = c(0.8, 0.85, 0.9,
    0.95, 1)),
  BootStrapType = list(classifier = c("Bayesian", "Bernoulli", "Poisson", "MVS", "No"),
    regression = c("Bayesian", "Bernoulli", "Poisson", "MVS", "No")),
  GrowPolicy = list(classifier = c("SymmetricTree", "Depthwise", "Lossguide"),
    regression = c("SymmetricTree", "Depthwise", "Lossguide"))
)
```

Arguments

data	Source training data. Do not include a column that has the class labels for the buckets as they are created internally.
TimeWeights	Supply a value that will be multiplied by he time trend value
TrainOnFull	Set to TRUE to use all data
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

TransformNumericColumns
: Transform numeric column inside the AutoCatBoostRegression() function

Methods
: Choose transformation methods

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
: The path to your folder where you want your model information saved

MetaDataPaths
: TA character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to Paths.

SaveModelObjects
: Set to TRUE to save the model objects to file in the folders listed in Paths

ReturnModelObjects
: TRUE to return 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

GridTune
: Set to TRUE if you want to grid tune the models

BaselineComparison
: = "default",

MaxModelsInGrid
: = 1L,

MaxRunsWithoutNewWinner
: = 20L,

MaxRunMinutes
: = 60L*60L,

Shuffles
: = 2L,

MetricPeriods
: = 25L,

Langevin
: TRUE or FALSE

DiffusionTemperature
: Default 10000

Trees
: Provide a named list to have different number of trees for each model. Trees = list("classifier" = seq(1000,2000,100), "regression" = seq(1000,2000,100))

Depth
: = seq(4L, 8L, 1L),

RandomStrength
: 1

BorderCount
: 128

LearningRate
: = seq(0.01,0.10,0.01),

L2_Leaf_Reg
: = seq(1.0, 10.0, 1.0),

RSM
: = c(0.80, 0.85, 0.90, 0.95, 1.0),

BootStrapType
: = c("Bayesian", "Bernoulli", "Poisson", "MVS", "No"),

GrowPolicy
: = c("SymmetricTree", "Depthwise", "Lossguide")

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 catboost-grid

Author(s)

Adrian Antico

See Also

Other Supervised Learning - Compound: AutoCatBoostSizeFreqDist(), AutoH2oDRFHurdleModel(), AutoH2oGBMHurdleModel(), AutoH2oGBMSizeFreqDist(), AutoXGBoostHurdleModel()

Examples

```
## Not run:
Output <- RemixAutoML::AutoCatBoostHurdleModel(

  # Operationalization
  task_type = "GPU",
  ModelID = "ModelTest",
  SaveModelObjects = FALSE,
  ReturnModelObjects = TRUE,

  # Data related args
  data = data,
  TimeWeights = NULL,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  Buckets = 0L,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  PrimaryDateColumn = NULL,
  IDcols = NULL,

  # Metadata args
  Paths = normalizePath("./"),
  MetaDataPaths = NULL,
  TransformNumericColumns = NULL,
  Methods =
     c("BoxCox", "Asinh", "Asin", "Log",
       "LogPlus1", "Logit", "YeoJohnson"),
  ClassWeights = NULL,
  SplitRatios = c(0.70, 0.20, 0.10),
  NumOfParDepPlots = 10L,

  # Grid tuning setup
  PassInGrid = NULL,
  GridTune = FALSE,
  BaselineComparison = "default",
  MaxModelsInGrid = 1L,
  MaxRunsWithoutNewWinner = 20L,
  MaxRunMinutes = 60L*60L,
```

```
  Shuffles = 2L,
  MetricPeriods = 25L,

  # Bandit grid args
  Langevin = FALSE,
  DiffusionTemperature = 10000,
  Trees = list("classifier" = seq(1000,2000,100),
               "regression" = seq(1000,2000,100)),
  Depth = list("classifier" = seq(6,10,1),
               "regression" = seq(6,10,1)),
  RandomStrength = list("classifier" = seq(1,10,1),
                        "regression" = seq(1,10,1)),
  BorderCount = list("classifier" = seq(32,256,16),
                     "regression" = seq(32,256,16)),
  LearningRate = list("classifier" = seq(0.01,0.25,0.01),
                      "regression" = seq(0.01,0.25,0.01)),
  L2_Leaf_Reg = list("classifier" = seq(3.0,10.0,1.0),
                     "regression" = seq(1.0,10.0,1.0)),
  RSM = list("classifier" = c(0.80, 0.85, 0.90, 0.95, 1.0),
             "regression" = c(0.80, 0.85, 0.90, 0.95, 1.0)),
  BootStrapType = list("classifier" = c("Bayesian", "Bernoulli", "Poisson", "MVS", "No"),
                       "regression" = c("Bayesian", "Bernoulli", "Poisson", "MVS", "No")),
  GrowPolicy = list("classifier" = c("SymmetricTree", "Depthwise", "Lossguide"),
                    "regression" = c("SymmetricTree", "Depthwise", "Lossguide")))

## End(Not run)
```

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,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  PrimaryDateColumn = NULL,
  ClassWeights = NULL,
```

```
  IDcols = NULL,
  task_type = "GPU",
  eval_metric = "MultiClassOneVsAll",
  loss_function = "MultiClassOneVsAll",
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  PassInGrid = NULL,
  GridTune = FALSE,
  MaxModelsInGrid = 10L,
  MaxRunsWithoutNewWinner = 20L,
  MaxRunMinutes = 24L * 60L,
  grid_eval_metric = "Accuracy",
  Shuffles = 1L,
  BaselineComparison = "default",
  MetricPeriods = 10L,
  Trees = 50L,
  Depth = 6,
  LearningRate = NULL,
  L2_Leaf_Reg = NULL,
  RandomStrength = 1,
  BorderCount = 128,
  RSM = NULL,
  BootStrapType = NULL,
  GrowPolicy = NULL
)
```

Arguments

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

`TrainOnFull` Set to TRUE to train on full data and skip over evaluation steps

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

eval_metric — Internal bandit metric. Select from 'MultiClass', 'MultiClassOneVsAll', 'AUC', 'TotalF1', 'MCC', 'Accuracy', 'HingeLoss', 'HammingLoss', 'ZeroOneLoss', 'Kappa', 'WKappa'

loss_function — Select from 'MultiClass' or 'MultiClassOneVsAll'

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

metadata_path — A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to model_path.

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)

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.

MaxModelsInGrid — Number of models to test from grid options.

MaxRunsWithoutNewWinner — A number

MaxRunMinutes — In minutes

grid_eval_metric — For evaluating models within grid tuning. Choices include, "accuracy", "microauc", "logloss"

Shuffles — Numeric. List a number to let the program know how many times you want to shuffle the grids for grid tuning

BaselineComparison — Set to either "default" or "best". Default is to compare each successive model build to the baseline model using max trees (from function args). Best makes the comparison to the current best model.

MetricPeriods — Number of trees to build before evaluating intermediate metrics. Default is 10L

Trees — Bandit grid partitioned. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the trees numbers you want to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(1000L, 10000L, 1000L)

Depth — Bandit grid partitioned. Number, or vector for depth to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(4L, 16L, 2L)

LearningRate — Bandit grid partitioned. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the LearningRate values to test. For running grid tuning, a NULL value supplied will mean these values are tested c(0.01,0.02,0.03,0.04)

L2_Leaf_Reg — Random testing. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the L2_Leaf_Reg values to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(1.0, 10.0, 1.0)

RandomStrength A multiplier of randomness added to split evaluations. Default value is 1 which adds no randomness.

BorderCount Number of splits for numerical features. Catboost defaults to 254 for CPU and 128 for GPU

RSM CPU only. Random testing. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the RSM values to test. For running grid tuning, a NULL value supplied will mean these values are tested c(0.80, 0.85, 0.90, 0.95, 1.0)

BootStrapType Random testing. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the BootStrapType values to test. For running grid tuning, a NULL value supplied will mean these values are tested c("Bayesian", "Bernoulli", "Poisson", "MVS", "No")

GrowPolicy Random testing. NULL, character, or vector for GrowPolicy to test. For grid tuning, supply a vector of values. For running grid tuning, a NULL value supplied will mean these values are tested c("SymmetricTree", "Depthwise", "Lossguide")

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 Automated Supervised Learning - Multiclass Classification: AutoH2oDRFMultiClass(), AutoH2oGAMMultiClass(), AutoH2oGBMMultiClass(), AutoH2oGLMMultiClass(), AutoH2oMLMultiClass(), AutoXGBoostMultiClass()

Examples

```
## Not run:
# Create some dummy correlated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 10000L,
  ID = 2L,
  ZIP = 0L,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = TRUE)

# Run function
TestModel <- RemixAutoML::AutoCatBoostMultiClass(

    # GPU or CPU and the number of available GPUs
    task_type = "GPU",

    # Metadata arguments:
    #   'ModelID' is used to create part of the file
    #       names generated when saving to file'
```

```
#   'model_path' is where the minimal model objects
#       for scoring will be stored
#   'ModelID' will be the name of the saved model object
#   'metadata_path' is where model evaluation and model
#       interpretation files are saved
#    objects saved to model_path if metadata_path is null
#    Saved objects include:
#    'ModelID_ValidationData.csv' is the supplied or generated
#       TestData with predicted values
#    'ModelID_ROC_Plot.png' and 'Model_ID_EvaluationPlot.png'
#        calibration plot
#    'ModelID_VariableImportance.csv' is the variable importance.
#        This won't be saved to file if GrowPolicy is either
#          "Depthwise" or "Lossguide" was used
#    'ModelID_ExperimentGrid.csv' if GridTune = TRUE.
#        Results of all model builds including parameter settings,
#          bandit probs, and grid IDs
#    'ModelID_EvaluationMetrics.csv' which contains all confusion
#           matrix measures across all thresholds
ModelID = "Test_Model_1",
model_path = normalizePath("./"),
metadata_path = file.path(normalizePath("./"),"R_Model_Testing"),
SaveModelObjects = FALSE,
ReturnModelObjects = TRUE,

# Data arguments:
#   'TrainOnFull' is to train a model with 100 percent of
#      your data.
#   That means no holdout data will be used for evaluation
#   If ValidationData and TestData are NULL and TrainOnFull
#      is FALSE then data will be split 70 20 10
#   'PrimaryDateColumn' is a date column in data that is
#      meaningful when sorted.
#    CatBoost categorical treatment is enhanced when supplied
#   'IDcols' are columns in your data that you don't use for
#      modeling but get returned with ValidationData
data = data,
TrainOnFull = FALSE,
ValidationData = NULL,
TestData = NULL,
TargetColumnName = "Adrian",
FeatureColNames = names(data)[!names(data) %chin%
  c("IDcol_1", "IDcol_2","Adrian")],
PrimaryDateColumn = NULL,
ClassWeights = c(1L,1L,1L,1L,1L),
IDcols = c("IDcol_1","IDcol_2"),

# Model evaluation:
#   'eval_metric' is the measure catboost uses when evaluting
#       on holdout data during its bandit style process
#   'loss_function' the loss function used in training optimization
eval_metric = "MCC",
loss_function = "MultiClassOneVsAll",
grid_eval_metric = "Accuracy",
MetricPeriods = 10L,

# Grid tuning arguments:
```

```
    #   'PassInGrid' is for retraining using a previous grid winning args
    #   'MaxModelsInGrid' is a cap on the number of models that will run
    #   'MaxRunsWithoutNewWinner' number of runs without a new winner
    #      before exiting grid tuning
    #   'MaxRunMinutes' is a cap on the number of minutes that will run
    #   'Shuffles' is the number of times you want the random grid
    #      arguments shuffled
    #   'BaselineComparison' default means to compare each model build
    #      with a default built of catboost using max(Trees)
    #   'MetricPeriods' is the number of trees built before evaluting
    #      holdoutdata internally. Used in finding actual Trees used.
    PassInGrid = NULL,
    GridTune = FALSE,
    MaxModelsInGrid = 100L,
    MaxRunsWithoutNewWinner = 20L,
    MaxRunMinutes = 24L*60L,
    Shuffles = 4L,
    BaselineComparison = "default",

    # Trees, Depth, and LearningRate used in the bandit grid tuning
    # Must set Trees to a single value if you are not grid tuning
    # The ones below can be set to NULL and the values in the example
    #   will be used
    # GrowPolicy is turned off for CPU runs
    # BootStrapType utilizes Poisson only for GPU and MVS only for CPU
    Trees = seq(100L, 500L, 50L),
    Depth = seq(4L, 8L, 1L),
    LearningRate = seq(0.01,0.10,0.01),
    L2_Leaf_Reg = seq(1.0, 10.0, 1.0),
    RandomStrength = 1,
    BorderCount = 254,
    RSM = c(0.80, 0.85, 0.90, 0.95, 1.0),
    BootStrapType = c("Bayesian", "Bernoulli", "Poisson", "MVS", "No"),
    GrowPolicy = c("SymmetricTree", "Depthwise", "Lossguide"))

# Output
TestModel$Model
TestModel$ValidationData
TestModel$EvaluationMetrics
TestModel$Evaluation
TestModel$VI_Plot
TestModel$VariableImportance
TestModel$InteractionImportance
TestModel$GridMetrics
TestModel$ColNames = Names,
TestModel$TargetLevels

## End(Not run)
```

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,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  Weights = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  PrimaryDateColumn = NULL,
  DummifyCols = FALSE,
  IDcols = NULL,
  TransformNumericColumns = NULL,
 Methods = c("BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson"),
  task_type = "GPU",
  NumGPUs = 1,
  eval_metric = "RMSE",
  eval_metric_value = 1.5,
  loss_function = "RMSE",
  loss_function_value = 1.5,
  model_path = NULL,
  metadata_path = NULL,
  SaveInfoToPDF = FALSE,
  ModelID = "FirstModel",
  NumOfParDepPlots = 0L,
  EvalPlots = TRUE,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  PassInGrid = NULL,
  GridTune = FALSE,
  MaxModelsInGrid = 10L,
  MaxRunsWithoutNewWinner = 20L,
  MaxRunMinutes = 24L * 60L,
  Shuffles = 1L,
  BaselineComparison = "default",
  MetricPeriods = 10L,
  langevin = FALSE,
  diffusion_temperature = 10000,
  Trees = 50L,
  Depth = 6,
  L2_Leaf_Reg = 3,
  RandomStrength = 1,
```

```
    BorderCount = 128,
    LearningRate = NULL,
    RSM = NULL,
    BootStrapType = NULL,
    GrowPolicy = NULL
  )
```

Arguments

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

`TrainOnFull` Set to TRUE to train on full data and skip over evaluation steps

`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.

`Weights` Weights vector for train.pool in catboost

`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

`DummifyCols` Logical. Will coerce to TRUE if loss_function or eval_metric is set to 'MultiRMSE'.

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

`TransformNumericColumns` Set to NULL to do nothing; otherwise supply the column names of numeric variables you want transformed

`Methods` Choose from "BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson". Function will determine if one cannot be used because of the underlying data.

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

`NumGPUs` Set to 1, 2, 3, etc.

`eval_metric` Select from 'RMSE', 'MAE', 'MAPE', 'R2', 'Poisson', 'MedianAbsoluteError', 'SMAPE', 'MSLE', 'NumErrors', 'FairLoss', 'Tweedie', 'Huber', 'LogLinQuantile', 'Quantile', 'Lq', 'Expectile'

`eval_metric_value` Used with the specified eval_metric. See https://catboost.ai/docs/concepts/loss-functions-regression.html

`loss_function` Used in model training for model fitting. 'MAPE', 'MAE', 'RMSE', 'Poisson', 'Tweedie', 'Huber', 'LogLinQuantile', 'Quantile', 'Lq', 'Expectile'

loss_function_value
: Used with the specified loss function if an associated value is required. 'Tweedie', 'Huber', 'LogLinQuantile', 'Quantile' 'Lq', 'Expectile'. See https://catboost.ai/docs/concepts/loss-functions-regression.html

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

metadata_path
: A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to model_path.

SaveInfoToPDF
: Set to TRUE to save modeling information to PDF. If model_path or metadata_path aren't defined then output will be saved to the working directory

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)

EvalPlots
: Defaults to TRUE. Set to FALSE to not generate and return these objects.

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)

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.

MaxModelsInGrid
: Number of models to test from grid options

MaxRunsWithoutNewWinner
: Number of models built before calling it quits

MaxRunMinutes
: Maximum number of minutes to let this run

Shuffles
: Number of times to randomize grid possibilities

BaselineComparison
: Set to either "default" or "best". Default is to compare each successive model build to the baseline model using max trees (from function args). Best makes the comparison to the current best model.

MetricPeriods
: Number of periods to use between Catboost evaluations

langevin
: Set to TRUE to enable

diffusion_temperature
: Defaults to 10000

Trees
: Bandit grid partitioned. The maximum number of trees you want in your models

Depth
: Bandit grid partitioned. Number, or vector for depth to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(4L, 16L, 2L)

L2_Leaf_Reg
: Random testing. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the L2_Leaf_Reg values to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(1.0, 10.0, 1.0)

RandomStrength
: A multiplier of randomness added to split evaluations. Default value is 1 which adds no randomness.

`BorderCount` Number of splits for numerical features. Catboost defaults to 254 for CPU and 128 for GPU

`LearningRate` Bandit grid partitioned. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the LearningRate values to test. For running grid tuning, a NULL value supplied will mean these values are tested c(0.01,0.02,0.03,0.04)

`RSM` CPU only. Random testing. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the RSM values to test. For running grid tuning, a NULL value supplied will mean these values are tested c(0.80, 0.85, 0.90, 0.95, 1.0)

`BootStrapType` Random testing. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the BootStrapType values to test. For running grid tuning, a NULL value supplied will mean these values are tested c("Bayesian", "Bernoulli", "Poisson", "MVS", "No")

`GrowPolicy` Random testing. NULL, character, or vector for GrowPolicy to test. For grid tuning, supply a vector of values. For running grid tuning, a NULL value supplied will mean these values are tested c("SymmetricTree", "Depthwise", "Lossguide")

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, catboostgrid, and a transformation details file.

Author(s)

Adrian Antico

See Also

Other Automated Supervised Learning - Regression: `AutoH2oDRFRegression()`, `AutoH2oGAMRegression()`, `AutoH2oGBMRegression()`, `AutoH2oGLMRegression()`, `AutoH2oMLRegression()`, `AutoXGBoostRegression()`

Examples

```
## Not run:
# Create some dummy correlated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 10000,
  ID = 2,
  ZIP = 0,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run function
TestModel <- RemixAutoML::AutoCatBoostRegression(

    # GPU or CPU and the number of available GPUs
    task_type = "GPU",
    NumGPUs = 1,
```

```
# Metadata arguments:
#   'ModelID' is used to create part of the file
#       names generated when saving to file'
#   'model_path' is where the minimal model objects
#       for scoring will be stored
#   'ModelID' will be the name of the saved model object
#   'metadata_path' is where model evaluation and model
#       interpretation files are saved
#    objects saved to model_path if metadata_path is null
#    Saved objects include:
#    'ModelID_ValidationData.csv' is the supplied or generated
#       TestData with predicted values
#    'ModelID_ROC_Plot.png' and 'Model_ID_EvaluationPlot.png'
#        calibration plot
#    'ModelID_VariableImportance.csv' is the variable importance.
#        This won't be saved to file if GrowPolicy is either
#          "Depthwise" or "Lossguide" was used
#    'ModelID_ExperimentGrid.csv' if GridTune = TRUE.
#        Results of all model builds including parameter settings,
#          bandit probs, and grid IDs
#    'ModelID_EvaluationMetrics.csv' which contains all confusion
#           matrix measures across all thresholds
ModelID = "Test_Model_1",
model_path = normalizePath("./"),
metadata_path = NULL,
SaveModelObjects = FALSE,
SaveInfoToPDF = FALSE,
ReturnModelObjects = TRUE,

# Data arguments:
#   'TrainOnFull' is to train a model with 100 percent of
#      your data.
#   That means no holdout data will be used for evaluation
#   If ValidationData and TestData are NULL and TrainOnFull
#      is FALSE then data will be split 70 20 10
#   'PrimaryDateColumn' is a date column in data that is
#      meaningful when sorted.
#    CatBoost categorical treatment is enhanced when supplied
#   'IDcols' are columns in your data that you don't use for
#      modeling but get returned with ValidationData
data = data,
TrainOnFull = FALSE,
ValidationData = NULL,
TestData = NULL,
Weights = NULL,
TargetColumnName = "Adrian",
FeatureColNames = names(data)[!names(data) %chin%
  c("IDcol_1", "IDcol_2","Adrian")],
PrimaryDateColumn = NULL,
DummifyCols = FALSE,
IDcols = c("IDcol_1","IDcol_2"),
TransformNumericColumns = "Adrian",
Methods = c("BoxCox", "Asinh", "Asin", "Log",
  "LogPlus1", "Logit", "YeoJohnson"),

# Model evaluation:
#   'eval_metric' is the measure catboost uses when evaluting
```

```
    #       on holdout data during its bandit style process
    #   'loss_function' the loss function used in training optimization
    #   'NumOfParDepPlots' Number of partial dependence calibration plots
    #       generated.
    #     A value of 3 will return plots for the top 3 variables based
    #       on variable importance
    #     Won't be returned if GrowPolicy is either "Depthwise" or
    #       "Lossguide" is used
    #     Can run the RemixAutoML::ParDepCalPlots() with the outputted
    #        ValidationData
    eval_metric = "RMSE",
    eval_metric_value = 1.5,
    loss_function = "RMSE",
    loss_function_value = 1.5,
    MetricPeriods = 10L,
    NumOfParDepPlots = ncol(data)-1L-2L,
    EvalPlots = TRUE,

    # Grid tuning arguments:
    #   'PassInGrid' is for retraining using a previous grid winning args
    #   'MaxModelsInGrid' is a cap on the number of models that will run
    #   'MaxRunsWithoutNewWinner' number of runs without a new winner
    #      before exiting grid tuning
    #   'MaxRunMinutes' is a cap on the number of minutes that will run
    #   'Shuffles' is the number of times you want the random grid
    #      arguments shuffled
    #   'BaselineComparison' default means to compare each model build
    #      with a default built of catboost using max(Trees)
    #   'MetricPeriods' is the number of trees built before evaluting
    #      holdoutdata internally. Used in finding actual Trees used.
    PassInGrid = NULL,
    GridTune = FALSE,
    MaxModelsInGrid = 100L,
    MaxRunsWithoutNewWinner = 100L,
    MaxRunMinutes = 60*60,
    Shuffles = 4L,
    BaselineComparison = "default",

    # Trees, Depth, and LearningRate used in the bandit grid tuning
    # Must set Trees to a single value if you are not grid tuning
    # The ones below can be set to NULL and the values in the example
    #   will be used
    # GrowPolicy is turned off for CPU runs
    # BootStrapType utilizes Poisson only for GPU and MVS only for CPU
    langevin = FALSE,
    diffusion_temperature = 10000,
    Trees = 1000,
    Depth = 6,
    L2_Leaf_Reg = 3.0,
    RandomStrength = 1,
    BorderCount = 128,
    LearningRate = seq(0.01,0.10,0.01),
    RSM = c(0.80, 0.85, 0.90, 0.95, 1.0),
    BootStrapType = c("Bayesian", "Bernoulli", "Poisson", "MVS", "No"),
    GrowPolicy = c("SymmetricTree", "Depthwise", "Lossguide"))

# Output
```

```
TestModel$Model
TestModel$ValidationData
TestModel$EvaluationPlot
TestModel$EvaluationBoxPlot
TestModel$EvaluationMetrics
TestModel$VariableImportance
TestModel$InteractionImportance
TestModel$ShapValuesDT
TestModel$VI_Plot
TestModel$PartialDependencePlots
TestModel$PartialDependenceBoxPlots
TestModel$GridList
TestModel$ColNames
TestModel$TransformationResults

## End(Not run)
```

`AutoCatBoostScoring` *AutoCatBoostScoring*

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,
  FactorLevelsList = NULL,
  IDcols = NULL,
  OneHot = FALSE,
  ReturnShapValues = FALSE,
  ModelObject = NULL,
  ModelPath = NULL,
  ModelID = NULL,
  ReturnFeatures = TRUE,
  MultiClassTargetLevels = NULL,
  TransformNumeric = FALSE,
  BackTransNumeric = FALSE,
  TargetColumnName = NULL,
  TransformationObject = NULL,
  TransID = NULL,
  TransPath = NULL,
  MDP_Impute = TRUE,
  MDP_CharToFactor = TRUE,
  MDP_RemoveDates = TRUE,
  MDP_MissFactor = "0",
  MDP_MissNum = -1,
```

```
  RemoveModel = FALSE
)
```

Arguments

`TargetType` Set this value to "regression", "classification", "multiclass", or "multiregression" 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

`FactorLevelsList` List of factors levels to DummifyDT()

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

`OneHot` Passsed to DummifyD

`ReturnShapValues` Set to TRUE to return a data.table of feature contributions to all predicted values generated

`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.

`MultiClassTargetLevels` For use with AutoCatBoostMultiClass(). If you saved model objects then this scoring function will locate the target levels file. If you did not save model objects, you can supply the target levels returned from AutoCatBoostMultiClass().

`TransformNumeric` Set to TRUE if you have features that were transformed automatically from an Auto__Regression() model AND you haven't already transformed them.

`BackTransNumeric` Set to TRUE to generate back-transformed predicted values. Also, if you return features, those will also be back-transformed.

`TargetColumnName` Input your target column name used in training if you are utilizing the transformation service

`TransformationObject` Set to NULL if you didn't use transformations or if you want the function to pull from the file output from the Auto__Regression() function. You can also supply the transformation data.table object with the transformation details versus having it pulled from file.

`TransID` Set to the ID used for saving the transformation data.table object or set it to the ModelID if you are pulling from file from a build with Auto__Regression().

`TransPath` Set the path file to the folder where your transformation data.table detail object is stored. If you used the Auto__Regression() to build, set it to the same path as ModelPath.

`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

`RemoveModel` Set to TRUE if you want the model removed immediately after scoring

Value

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

Author(s)

Adrian Antico

See Also

Other Automated Model Scoring: AutoH2OMLScoring(), AutoH2OModeler(), AutoHurdleScoring(), AutoXGBoostScoring(), IntermittentDemandScoringDataGenerator()

Examples

```
## Not run:

# Create some dummy correlated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 10000,
  ID = 2,
  ZIP = 0,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Train a Multiple Regression Model (two target variables)
TestModel <- RemixAutoML::AutoCatBoostRegression(

  # GPU or CPU and the number of available GPUs
  task_type = "GPU",
  NumGPUs = 1,

  # Metadata arguments
  ModelID = "Test_Model_1",
  model_path = normalizePath("./"),
  metadata_path = NULL,
  SaveModelObjects = FALSE,
  ReturnModelObjects = TRUE,
```

```
  # Data arguments
  data = data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  Weights = NULL,
  DummifyCols = FALSE,
  TargetColumnName = c("Adrian","Independent_Variable1"),
  FeatureColNames = names(data)[!names(data) %in%
    c("IDcol_1","IDcol_2","Adrian")],
  PrimaryDateColumn = NULL,
  IDcols = c("IDcol_1","IDcol_2"),
  TransformNumericColumns = NULL,
  Methods = c("BoxCox","Asinh","Asin","Log","LogPlus1",
    "Logit","YeoJohnson"),

  # Model evaluation
  eval_metric = "MultiRMSE",
  eval_metric_value = 1.5,
  loss_function = "MultiRMSE",
  loss_function_value = 1.5,
  MetricPeriods = 10L,
  NumOfParDepPlots = ncol(data)-1L-2L,
  EvalPlots = TRUE,

  # Grid tuning
  PassInGrid = NULL,
  GridTune = FALSE,
  MaxModelsInGrid = 100L,
  MaxRunsWithoutNewWinner = 100L,
  MaxRunMinutes = 60*60,
  Shuffles = 4L,
  BaselineComparison = "default",

  # ML Args
  langevin = TRUE,
  diffusion_temperature = 10000,
  Trees = 250,
  Depth = 6,
  L2_Leaf_Reg = 3.0,
  RandomStrength = 1,
  BorderCount = 128,
  LearningRate = seq(0.01,0.10,0.01),
  RSM = c(0.80, 0.85, 0.90, 0.95, 1.0),
  BootStrapType = c("Bayesian","Bernoulli","Poisson","MVS","No"),
  GrowPolicy = c("SymmetricTree", "Depthwise", "Lossguide"))

# Output
TestModel$Model
TestModel$ValidationData
TestModel$EvaluationPlot
TestModel$EvaluationBoxPlot
TestModel$EvaluationMetrics
TestModel$VariableImportance
TestModel$InteractionImportance
TestModel$ShapValuesDT
TestModel$VI_Plot
```

```
TestModel$PartialDependencePlots
TestModel$PartialDependenceBoxPlots
TestModel$GridList
TestModel$ColNames
TestModel$TransformationResults

# Score a multiple regression model
Preds <- RemixAutoML::AutoCatBoostScoring(
  TargetType = "multiregression",
  ScoringData = data,
  FeatureColumnNames = names(data)[!names(data) %in%
    c("IDcol_1", "IDcol_2","Adrian")],
  FactorLevelsList = TestModel$FactorLevelsList,
  IDcols = c("IDcol_1","IDcol_2"),
  OneHot = FALSE,
  ReturnShapValues = TRUE,
  ModelObject = TestModel$Model,
  ModelPath = NULL, #normalizePath("./"),
  ModelID = "Test_Model_1",
  ReturnFeatures = TRUE,
  MultiClassTargetLevels = NULL,
  TransformNumeric = FALSE,
  BackTransNumeric = FALSE,
  TargetColumnName = NULL,
  TransformationObject = NULL,
  TransID = NULL,
  TransPath = NULL,
  MDP_Impute = TRUE,
  MDP_CharToFactor = TRUE,
  MDP_RemoveDates = TRUE,
  MDP_MissFactor = "0",
  MDP_MissNum = -1,
  RemoveModel = FALSE)

## End(Not run)
```

AutoCatBoostSizeFreqDist

AutoCatBoostSizeFreqDist

Description

AutoCatBoostSizeFreqDist for building size and frequency distributions via quantile regressions. Size (or severity) and frequency (or count) quantile regressions are build. Use this with the AutoQuantileGibbsSampler function to simulate the joint distribution.

Usage

```
AutoCatBoostSizeFreqDist(
  CountData = NULL,
  SizeData = NULL,
  CountQuantiles = seq(0.1, 0.9, 0.1),
  SizeQuantiles = seq(0.1, 0.9, 0.1),
  AutoTransform = TRUE,
```

```
  DataPartitionRatios = c(0.75, 0.2, 0.05),
  StratifyColumnNames = NULL,
  NTrees = 1500,
  TaskType = "GPU",
  EvalMetric = "Quantile",
  GridTune = FALSE,
  GridEvalMetric = "mae",
  CountTargetColumnName = NULL,
  SizeTargetColumnName = NULL,
  CountFeatureColNames = NULL,
  SizeFeatureColNames = NULL,
  CountIDcols = NULL,
  SizeIDcols = NULL,
  ModelIDs = c("CountModel", "SizeModel"),
  MaxModelsGrid = 5,
  ModelPath = NULL,
  MetaDataPath = NULL,
  NumOfParDepPlots = 0
)
```

Arguments

`CountData` This is your CountData generated from the IntermittentDemandBootStrapper() function

`SizeData` This is your SizeData generated from the IntermittentDemandBootStrapper() function

`CountQuantiles` The default are deciles, i.e. seq(0.10,0.90,0.10). More granularity the better, but it will take longer to run.

`SizeQuantiles` The default are deciles, i.e. seq(0.10,0.90,0.10). More granularity the better, but it will take longer to run.

`AutoTransform` Set to FALSE not to have the your target variables automatically transformed for the best normalization.

`DataPartitionRatios` The default is c(0.75,0.20,0.05). With CatBoost, you should allocate a decent amount to the validation data (second input). Three inputs are required.

`StratifyColumnNames` Specify grouping variables to stratify by

`NTrees` Default is 1500. If the best model utilizes all trees, you should consider increasing the argument.

`TaskType` The default is set to "GPU". If you do not have a GPU, set it to "CPU".

`EvalMetric` Set to "Quantile". Alternative quantile methods may become available in the future.

`GridTune` The default is set to FALSE. If you set to TRUE, make sure to specify MaxModelsGrid to a number greater than 1.

`GridEvalMetric` The default is set to "mae". Choose from 'poisson', 'mae', 'mape', 'mse', 'msle', 'kl', 'cs', 'r2'.

`CountTargetColumnName` Column names or column numbers

`SizeTargetColumnName` Column names or column numbers

`CountFeatureColNames`
: Column names or column numbers

`SizeFeatureColNames`
: Column names or column numbers

`CountIDcols`
: Column names or column numbers

`SizeIDcols`
: Column names or column numbers

`ModelIDs`
: A two element character vector. E.g. c("CountModel","SizeModel")

`MaxModelsGrid`
: Set to a number greater than 1 if GridTune is set to TRUE

`ModelPath`
: This path file is where all your models will be stored. If you leave MetaDataPath NULL, the evaluation metadata will also be stored here. If you leave this NULL, the function will not run.

`MetaDataPath`
: A separate path to store the model metadata for evaluation.

`NumOfParDepPlots`
: Set to a number greater than or equal to 1 to see the relationships between your features and targets.

Value

This function does not return anything. It can only store your models and model evaluation metadata to file.

Author(s)

Adrian Antico

See Also

Other Supervised Learning - Compound: `AutoCatBoostHurdleModel()`, `AutoH2oDRFHurdleModel()`, `AutoH2oGBMHurdleModel()`, `AutoH2oGBMSizeFreqDist()`, `AutoXGBoostHurdleModel()`

Examples

```
## Not run:
AutoCatBoostSizeFreqDist(
  CountData = CountData,
  SizeData = SizeData,
  CountQuantiles = seq(0.10,0.90,0.10),
  SizeQuantiles = seq(0.10,0.90,0.10),
  AutoTransform = TRUE,
  DataPartitionRatios = c(0.75,0.20,0.05),
  StratifyColumnNames = NULL,
  NTrees = 1500,
  TaskType = "GPU",
  EvalMetric = "Quantile",
  GridTune = FALSE,
  GridEvalMetric = "mae",
  CountTargetColumnName = "Counts",
  SizeTargetColumnName = "Target_qty",
  CountFeatureColNames = 2:ncol(CountData),
  SizeFeatureColNames = 2:ncol(SizeData),
  CountIDcols = NULL,
  SizeIDcols = NULL,
  ModelIDs = c("CountModel","SizeModel"),
```

```
    MaxModelsGrid = 5,
    ModelPath = getwd(),
    MetaDataPath = paste0(getwd(),"/ModelMetaData"),
    NumOfParDepPlots = 1)

## End(Not run)
```

AutoCatBoostVectorCARMA
AutoCatBoostVectorCARMA

Description

AutoCatBoostVectorCARMA Multiple Regression, Mutlivariate Forecasting with calendar variables, Holiday counts, holiday lags, holiday moving averages, differencing, transformations, interaction-based categorical encoding using target variable and features to generate various time-based aggregated lags, moving averages, moving standard deviations, moving skewness, moving kurtosis, moving quantiles, parallelized interaction-based fourier pairs by grouping variables, and Trend Variables.

Usage

```
AutoCatBoostVectorCARMA(
  data,
  TimeWeights = NULL,
  NonNegativePred = FALSE,
  RoundPreds = FALSE,
  TrainOnFull = FALSE,
  TargetColumnName = "Target",
  DateColumnName = "DateTime",
  HierarchGroups = NULL,
  GroupVariables = NULL,
  FC_Periods = 30,
  TimeUnit = "week",
  TimeGroups = c("weeks", "months"),
  NumOfParDepPlots = 10L,
  TargetTransformation = FALSE,
  Methods = c("BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson"),
  AnomalyDetection = NULL,
  XREGS = NULL,
  Lags = c(1L:5L),
  MA_Periods = c(2L:5L),
  SD_Periods = NULL,
  Skew_Periods = NULL,
  Kurt_Periods = NULL,
  Quantile_Periods = NULL,
  Quantiles_Selected = c("q5", "q95"),
  Difference = TRUE,
  FourierTerms = 6L,
  CalendarVariables = c("second", "minute", "hour", "wday", "mday", "yday", "week",
    "isoweek", "month", "quarter", "year"),
  HolidayVariable = c("USPublicHolidays", "EasterGroup", "ChristmasGroup",
```

```
    "OtherEcclesticalFeasts"),
  HolidayLags = 1L,
  HolidayMovingAverages = 1L:2L,
  TimeTrendVariable = FALSE,
  ZeroPadSeries = NULL,
  DataTruncate = FALSE,
  SplitRatios = c(0.7, 0.2, 0.1),
  TaskType = "GPU",
  NumGPU = 1,
  EvalMetric = "RMSE",
  EvalMetricValue = 1.5,
  LossFunction = "RMSE",
  LossFunctionValue = 1.5,
  GridTune = FALSE,
  PassInGrid = NULL,
  ModelCount = 100,
  MaxRunsWithoutNewWinner = 50,
  MaxRunMinutes = 24L * 60L,
  Langevin = FALSE,
  DiffusionTemperature = 10000,
  NTrees = 1000,
  L2_Leaf_Reg = 3,
  RandomStrength = 1,
  BorderCount = 254,
  Depth = 6,
  BootStrapType = c("Bayesian", "Bernoulli", "Poisson", "MVS", "No"),
  PartitionType = "timeseries",
  Timer = TRUE,
  DebugMode = FALSE
)
```

Arguments

`data`	Supply your full series data set here
`TimeWeights`	NULL or a value.
`NonNegativePred`	TRUE or FALSE
`RoundPreds`	Rounding predictions to an integer value. TRUE or FALSE. Defaults to FALSE
`TrainOnFull`	Set to TRUE to train on full data
`TargetColumnName`	List the column names of your target variables column. E.g. c("Target1","Target2", ..., "TargetN")
`DateColumnName`	List the column name of your date column. E.g. "DateTime"
`HierarchGroups`	Vector of hierachy categorical columns.
`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. "1min", "5min", "10min", "15min", "30min", "hour", "day", "week", "month", "quarter", "year".

TimeGroups: Select time aggregations for adding various time aggregated GDL features.

NumOfParDepPlots: Supply a number for the number of partial dependence plots you want returned

TargetTransformation: Run AutoTransformationCreate() to find best transformation for the target variable. Tests YeoJohnson, BoxCox, and Asigh (also Asin and Logit for proportion target variables).

Methods: Transformation options to test which include "BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson"

AnomalyDetection: NULL for not using the service. Other, provide a list, e.g. AnomalyDetection = list("tstat_high" = 4, tstat_low = -4)

XREGS: Additional data to use for model development and forecasting. Data needs to be a complete series which means both the historical and forward looking values over the specified forecast window needs to be supplied.

Lags: Select the periods for all lag variables you want to create. E.g. c(1:5,52)

MA_Periods: Select the periods for all moving average variables you want to create. E.g. c(1:5,52)

SD_Periods: Select the periods for all moving standard deviation variables you want to create. E.g. c(1:5,52)

Skew_Periods: Select the periods for all moving skewness variables you want to create. E.g. c(1:5,52)

Kurt_Periods: Select the periods for all moving kurtosis variables you want to create. E.g. c(1:5,52)

Quantile_Periods: Select the periods for all moving quantiles variables you want to create. E.g. c(1:5,52)

Quantiles_Selected: Select from the following "q5", "q10", "q15", "q20", "q25", "q30", "q35", "q40", "q45", "q50", "q55", "q60", "q65", "q70", "q75", "q80", "q85", "q90", "q95"

Difference: Puts the I in ARIMA for single series and grouped series.

FourierTerms: Set to the max number of pairs. E.g. 2 means to generate two pairs for by each group level and interations if hierarchy is enabled.

CalendarVariables: NULL, or select from "second", "minute", "hour", "wday", "mday", "yday", "week", "isoweek", "month", "quarter", "year"

HolidayVariable: NULL, or select from "USPublicHolidays", "EasterGroup", "ChristmasGroup", "OtherEcclesticalFeasts"

HolidayLags: Number of lags to build off of the holiday count variable.

HolidayMovingAverages: Number of moving averages to build off of the holiday count variable.

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.

ZeroPadSeries	Set to "all", "inner", or NULL. See TimeSeriesFill for explanation
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	Has to CPU for now. If catboost makes GPU available for "MultiRMSE" then it will be enabled. If you set to GPU the function will coerce it back to CPU.
NumGPU	Defaults to 1. If CPU is set this argument will be ignored.
EvalMetric	"MultiRMSE" only. If catboost updates this I'll add more later
EvalMetricValue	Placeholder for later
LossFunction	"MultiRMSE" only. If catboost updates this I'll add more later
LossFunctionValue	Placeholder for later
GridTune	Set to TRUE to run a grid tune
PassInGrid	Defaults to NULL
ModelCount	Set the number of models to try in the grid tune
MaxRunsWithoutNewWinner	Default is 50
MaxRunMinutes	Default is 60*60
Langevin	TRUE or FALSE
DiffusionTemperature	Default value of 10000
NTrees	Select the number of trees you want to have built to train the model
L2_Leaf_Reg	l2 reg parameter
RandomStrength	Default is 1
BorderCount	Default is 254
Depth	Depth of catboost model
BootStrapType	Select from Catboost list
PartitionType	Select "random" for random data partitioning "timeseries" for partitioning by time frames
Timer	Set to FALSE to turn off the updating print statements for progress
DebugMode	Defaults to FALSE. Set to TRUE to get a print statement of each high level comment in function

Value

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

Author(s)

Adrian Antico

See Also

Other Automated Panel Data Forecasting: AutoCatBoostCARMA(), AutoCatBoostHurdleCARMA(), AutoH2OCARMA(), AutoXGBoostCARMA()

Examples

```
## Not run:
# Two group variables and xregs

# Load Walmart Data from Dropbox----
data <- data.table::fread(
 "https://www.dropbox.com/s/2str3ek4f4cheqi/walmart_train.csv?dl=1")

# Filter out zeros
data <- data[Weekly_Sales != 0]

# Subset for Stores / Departments With Full Series
data <- data[, Counts := .N, by = c("Store","Dept")][Counts == 143][
 , Counts := NULL]

# Subset Columns (remove IsHoliday column)----
keep <- c("Store","Dept","Date","Weekly_Sales")
data <- data[, ..keep]
data <- data[Store %in% c(1,2)]
xregs <- data.table::copy(data)
xregs[, GroupVar := do.call(paste, c(.SD, sep = " ")), .SDcols = c("Store","Dept")]
xregs[, c("Store","Dept") := NULL]
data.table::setnames(xregs, "Weekly_Sales", "Other")
xregs[, Other := jitter(Other, factor = 25)]
data <- data[as.Date(Date) < as.Date('2012-09-28')]

# Vector CARMA testing
data[, Weekly_Profit := Weekly_Sales * 0.75]

# Build forecast
CatBoostResults <- RemixAutoML::AutoCatBoostVectorCARMA(

  # data args
  data = data, # TwoGroup_Data,
  TimeWeights = NULL,
  TargetColumnName = c("Weekly_Sales","Weekly_Profit"),
  DateColumnName = "Date",
  HierarchGroups = NULL,
  GroupVariables = c("Store","Dept"),
  TimeUnit = "weeks",
  TimeGroups = c("weeks","months"),

  # Production args
  TrainOnFull = TRUE,
  SplitRatios = c(1 - 10 / 138, 10 / 138),
  PartitionType = "random",
  FC_Periods = 4,
  Timer = TRUE,
  DebugMode = TRUE,

  # Target transformations
```

```
  TargetTransformation = TRUE,
  Methods = c("BoxCox", "Asinh", "Asin", "Log",
              "LogPlus1", "Logit", "YeoJohnson"),
  Difference = FALSE,
  NonNegativePred = FALSE,
  RoundPreds = FALSE,

  # Date features
  CalendarVariables = c("week", "month", "quarter"),
  HolidayVariable = c("USPublicHolidays",
                      "EasterGroup",
                      "ChristmasGroup","OtherEcclesticalFeasts"),
  HolidayLags = 1,
  HolidayMovingAverages = 1:2,

  # Time series features
  Lags = list("weeks" = seq(2L, 10L, 2L),
              "months" = c(1:3)),
  MA_Periods = list("weeks" = seq(2L, 10L, 2L),
                    "months" = c(2,3)),
  SD_Periods = NULL,
  Skew_Periods = NULL,
  Kurt_Periods = NULL,
  Quantile_Periods = NULL,
  Quantiles_Selected = c("q5","q95"),

  # Bonus features
  AnomalyDetection = NULL,
  XREGS = xregs,
  FourierTerms = 2,
  TimeTrendVariable = TRUE,
  ZeroPadSeries = NULL,
  DataTruncate = FALSE,

  # ML Args
  NumOfParDepPlots = 100L,
  EvalMetric = "MultiRMSE",
  EvalMetricValue = 1.5,
  LossFunction = "MultiRMSE",
  LossFunctionValue = 1.5,
  GridTune = FALSE,
  PassInGrid = NULL,
  ModelCount = 5,
  TaskType = "GPU",
  NumGPU = 1,
  MaxRunsWithoutNewWinner = 50,
  MaxRunMinutes = 60*60,
  Langevin = FALSE,
  DiffusionTemperature = 10000,
  NTrees = 2500,
  L2_Leaf_Reg = 3.0,
  RandomStrength = 1,
  BorderCount = 254,
  BootStrapType = c("Bayesian", "Bernoulli", "Poisson", "MVS", "No"),
  Depth = 6)

## End(Not run)
```

AutoDataDictionaries *AutoDataDictionaries*

Description

AutoDataDictionaries is a function to return data dictionary data in table form

Usage

```
AutoDataDictionaries(
  Type = "sqlserver",
  DBConnection,
  DDType = 1L,
  Query = NULL,
  ASIS = FALSE,
  CloseChannel = TRUE
)
```

Arguments

Type	= "sqlserver" is currently the only system supported
DBConnection	This is a RODBC connection object for sql server
DDType	Select from 1 - 6 based on this article
Query	Supply a query
ASIS	Set to TRUE to pull in values without coercing types
CloseChannel	Set to TRUE to disconnect

Author(s)

Adrian Antico

See Also

Other Data Wrangling: ColumnSubsetDataTable(), DataDisplayMeta(), FakeDataGenerator(), FullFactorialCatFeatures(), SQL_ClearTable(), SQL_DropTable(), SQL_Query_Push(), SQL_Query(), SQL_SaveTable(), SQL_Server_DBConnection(), SQL_UpdateTable(), TimeSeriesMelt()

AutoDataPartition *AutoDataPartition*

Description

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

Usage

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

`StratifyNumericTarget` Supply a column name that is numeric. Use for "random" PartitionType, you can stratify your numeric variable by splitting up based on percRank to ensure a proper allocation of extreme values in your created data sets.

`StratTargetPrecision` For "random" PartitionType and when StratifyNumericTarget is not null, precision will be the number of decimals used in the percentile calculation. If you supply a value of 1, deciles will be used. For a value of 2, percentiles will be used. Larger values are supported.

`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: AutoHierarchicalFourier(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), ContinuousTimeDataGen CreateCalendarVariables(), CreateHolidayVariables(), DT_GDL_Feature_Engineering(), DummifyDT(), H2oAutoencoder(), ModelDataPrep(), Partial_DT_GDL_Feature_Engineering(), TimeSeriesFill()

Examples

```
# Create fake data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000,
  ID = 2,
  ZIP = 0,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run data partitioning function
dataSets <- RemixAutoML::AutoDataPartition(
  data,
  NumDataSets = 3L,
  Ratios = c(0.70,0.20,0.10),
  PartitionType = "random",
  StratifyColumnNames = NULL,
  StratifyNumericTarget = NULL,
  StratTargetPrecision = 1L,
  TimeColumnName = NULL)

# Collect data
TrainData <- dataSets$TrainData
ValidationData <- dataSets$ValidationData
TestData <- dataSets$TestData
```

AutoFourierFeatures *AutoFourierFeatures*

Description

#' AutoFourierFeatures

Usage

```
AutoFourierFeatures(
  data,
  FourierPairs = NULL,
  FCPeriods = NULL,
  Time_Unit = NULL,
  TargetColumn = NULL,
  DateColumn = NULL,
  GroupVariable = NULL,
```

```
  xregs = NonGroupDateNames
)
```

Arguments

data	The source data
FourierPairs	A number indicating the max number of fourier pairs that will be built
FCPeriods	Number of periods
Time_Unit	Agg level
TargetColumn	The name of your target column
DateColumn	The name of your date column
GroupVariable	The name of your group variable
xregs	Extra data to merge in

Author(s)

Adrian Antico

See Also

Other Feature Engineering Helper: ID_BuildTrainDataSets(), ID_MetadataGenerator(), ID_TrainingDataGenera
ID_TrainingDataGenerator()

AutoH2OCARMA *AutoH2OCARMA*

Description

AutoH2OCARMA Automated Panel Data and Time Series Forecasting using H2O algorithms, Calendar and Holiday variables, ARIMA features, Fouier variables, time trend, and transformations.

Usage

```
AutoH2OCARMA(
  AlgoType = "drf",
  ExcludeAlgos = "XGBoost",
  data,
  NonNegativePred = FALSE,
  RoundPreds = FALSE,
  TrainOnFull = FALSE,
  TargetColumnName = "Target",
  DateColumnName = "DateTime",
  HierarchGroups = NULL,
  GroupVariables = NULL,
  FC_Periods = 30,
  TimeUnit = "week",
  TimeGroups = c("weeks", "months"),
  TargetTransformation = FALSE,
 Methods = c("BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson"),
  XREGS = NULL,
```

```
  Lags = c(1:5),
  MA_Periods = c(1:5),
  SD_Periods = NULL,
  Skew_Periods = NULL,
  Kurt_Periods = NULL,
  Quantile_Periods = NULL,
  Quantiles_Selected = NULL,
  AnomalyDetection = NULL,
  Difference = TRUE,
  FourierTerms = 6,
  CalendarVariables = c("second", "minute", "hour", "wday", "mday", "yday", "week",
    "isoweek", "month", "quarter", "year"),
  HolidayVariable = c("USPublicHolidays", "EasterGroup", "ChristmasGroup",
    "OtherEcclesticalFeasts"),
  HolidayLags = 1,
  HolidayMovingAverages = 1:2,
  TimeTrendVariable = FALSE,
  DataTruncate = FALSE,
  ZeroPadSeries = NULL,
  SplitRatios = c(0.7, 0.2, 0.1),
  EvalMetric = "MAE",
  GridTune = FALSE,
  ModelCount = 1,
  NTrees = 1000,
  PartitionType = "timeseries",
  MaxMem = "32G",
  NThreads = max(1, parallel::detectCores() - 2),
  Timer = TRUE,
  DebugMode = FALSE
)
```

Arguments

AlgoType	Select from "dfr" for RandomForecast, "gbm" for gradient boosting, "glm" for generalized linear model, "automl" for H2O's AutoML algo, and "gam" for H2O's Generalized Additive Model.
ExcludeAlgos	For use when AlgoType = "AutoML". Selections include "DRF","GLM","XGBoost","GBM","DeepL and "Stacke-dEnsemble"
data	Supply your full series data set here
NonNegativePred	TRUE or FALSE
RoundPreds	Rounding predictions to an integer value. TRUE or FALSE. Defaults to FALSE
TrainOnFull	Set to TRUE to train on full data
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"
HierarchGroups	Vector of hierachy categorical columns.
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. "1min", "5min", "10min", "15min", "30min", "hour", "day", "week", "month", "quarter", "year".

TimeGroups
: Select time aggregations for adding various time aggregated GDL features.

TargetTransformation
: Run AutoTransformationCreate() to find best transformation for the target variable. Tests YeoJohnson, BoxCox, and Asigh (also Asin and Logit for proportion target variables).

Methods
: Choose from "BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson". Function will determine if one cannot be used because of the underlying data.

XREGS
: Additional data to use for model development and forecasting. Data needs to be a complete series which means both the historical and forward looking values over the specified forecast window needs to be supplied.

Lags
: Select the periods for all lag variables you want to create. E.g. c(1:5,52)

MA_Periods
: Select the periods for all moving average variables you want to create. E.g. c(1:5,52)

SD_Periods
: Select the periods for all moving standard deviation variables you want to create. E.g. c(1:5,52)

Skew_Periods
: Select the periods for all moving skewness variables you want to create. E.g. c(1:5,52)

Kurt_Periods
: Select the periods for all moving kurtosis variables you want to create. E.g. c(1:5,52)

Quantile_Periods
: Select the periods for all moving quantiles variables you want to create. E.g. c(1:5,52)

Quantiles_Selected
: Select from the following c("q5","q10","q15","q20","q25","q30","q35","q40","q45","q50","q55","q6

AnomalyDetection
: NULL for not using the service. Other, provide a list, e.g. AnomalyDetection = list("tstat_high" = 4, tstat_low = -4)

Difference
: Puts the I in ARIMA for single series and grouped series.

FourierTerms
: Set to the max number of pairs. E.g. 2 means to generate two pairs for by each group level and interations if hierarchy is enabled.

CalendarVariables
: NULL, or select from "second", "minute", "hour", "wday", "mday", "yday", "week", "isoweek", "month", "quarter", "year"

HolidayVariable
: NULL, or select from "USPublicHolidays", "EasterGroup", "ChristmasGroup", "OtherEcclesticalFeasts"

HolidayLags
: Number of lags to build off of the holiday count variable.

HolidayMovingAverages
: Number of moving averages to build off of the holiday count variable.

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
ZeroPadSeries	Set to "all", "inner", or NULL. See TimeSeriesFill for explanation
SplitRatios	E.g c(0.7,0.2,0.1) for train, validation, and test sets
EvalMetric	Select from "RMSE", "MAE", "MAPE", "Poisson", "Quantile", "LogLinQuantile", "Lq", "NumErrors", "SMAPE", "R2", "MSLE", "MedianAbsoluteError"
GridTune	Set to TRUE to run a grid tune
ModelCount	Set the number of models to try in the grid tune
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
MaxMem	Set to the maximum amount of memory you want to allow for running this function. Default is "32G".
NThreads	Set to the number of threads you want to dedicate to this function.
Timer	Set to FALSE to turn off the updating print statements for progress
DebugMode	Defaults to FALSE. Set to TRUE to get a print statement of each high level comment in function

Value

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

Author(s)

Adrian Antico

See Also

Other Automated Panel Data Forecasting: AutoCatBoostCARMA(), AutoCatBoostHurdleCARMA(), AutoCatBoostVectorCARMA(), AutoXGBoostCARMA()

Examples

```
## Not run:

# Load Walmart Data from Dropbox----
data <- data.table::fread(
  "https://www.dropbox.com/s/2str3ek4f4cheqi/walmart_train.csv?dl=1")

# Subset for Stores / Departments With Full Series
data <- data[, Counts := .N, by = c("Store","Dept")][Counts == 143][
  , Counts := NULL]

# Subset Columns (remove IsHoliday column)----
keep <- c("Store","Dept","Date","Weekly_Sales")
data <- data[, ..keep]
data <- data[Store == 1][, Store := NULL]
xregs <- data.table::copy(data)
```

```
data.table::setnames(xregs, "Dept", "GroupVar")
data.table::setnames(xregs, "Weekly_Sales", "Other")
data <- data[as.Date(Date) < as.Date('2012-09-28')]

# Build forecast
Results <- RemixAutoML::AutoH2OCARMA(

  # Data Artifacts
  AlgoType = "drf",
  ExcludeAlgos = NULL,
  data = data,
  TargetColumnName = "Weekly_Sales",
  DateColumnName = "Date",
  HierarchGroups = NULL,
  GroupVariables = c("Dept"),
  TimeUnit = "week",
  TimeGroups = c("weeks","months"),

  # Data Wrangling Features
  ZeroPadSeries = NULL,
  DataTruncate = FALSE,
  SplitRatios = c(1 - 10 / 138, 10 / 138),
  PartitionType = "random",

  # Productionize
  FC_Periods = 4L,
  TrainOnFull = FALSE,
  EvalMetric = "RMSE",
  GridTune = FALSE,
  ModelCount = 5,
  MaxMem = "28G",
  NThreads = parallel::detectCores(),
  Timer = TRUE,

  # Target Transformations
  TargetTransformation = FALSE,
  Methods = c("BoxCox", "Asinh", "Asin", "Log",
    "LogPlus1", "Logit", "YeoJohnson"),
  Difference = FALSE,
  NonNegativePred = FALSE,
  RoundPreds = FALSE,

  # Features
  AnomalyDetection = NULL,
  HolidayLags = 1:7,
  HolidayMovingAverages = 2:7,
  Lags = list("weeks" = c(1:4), "months" = c(1:3)),
  MA_Periods = list("weeks" = c(2:8), "months" = c(6:12)),
  SD_Periods = NULL,
  Skew_Periods = NULL,
  Kurt_Periods = NULL,
  Quantile_Periods = NULL,
  Quantiles_Selected = NULL,
  XREGS = NULL,
  FourierTerms = 2L,
  CalendarVariables = c("week", "month", "quarter", "year"),
  HolidayVariable = c("USPublicHolidays","EasterGroup",
```

```
    "ChristmasGroup","OtherEcclesticalFeasts"),
  TimeTrendVariable = TRUE,
  NTrees = 1000L,
  DebugMode = TRUE)

UpdateMetrics <-
  Results$ModelInformation$EvaluationMetrics[
    Metric == "MSE", MetricValue := sqrt(MetricValue)]
print(UpdateMetrics)

# Get final number of trees actually used
Results$Model@model$model_summary$number_of_internal_trees

# Inspect performance
Results$ModelInformation$EvaluationMetricsByGroup[order(-R2_Metric)]
Results$ModelInformation$EvaluationMetricsByGroup[order(MAE_Metric)]
Results$ModelInformation$EvaluationMetricsByGroup[order(MSE_Metric)]
Results$ModelInformation$EvaluationMetricsByGroup[order(MAPE_Metric)]

## End(Not run)
```

`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,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  eval_metric = "auc",
  Trees = 50L,
  GridTune = FALSE,
  MaxMem = "32G",
  NThreads = max(1, parallel::detectCores() - 2),
  MaxModelsInGrid = 2,
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
```

```
  NumOfParDepPlots = 3L,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = FALSE,
  HurdleModel = FALSE
)
```

Arguments

data	This is your data set for training and testing your model
TrainOnFull	Set to TRUE to train on full data
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"
NThreads	Set the number of threads you want to dedicate to the model building
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
metadata_path	A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to model_path.
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
H2OShutdown	Set to TRUE to shutdown H2O after running the function
HurdleModel	Leave it set to FALSE

Value

Saves to file and returned in list: VariableImportance.csv, Model, ValidationData.csv, Evalution-Plot.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 Automated Supervised Learning - Binary Classification: AutoCatBoostClassifier(), AutoH2oGAMClassifier(), AutoH2oGBMClassifier(), AutoH2oGLMClassifier(), AutoH2oMLClassifier(), AutoXGBoostClassifier()

Examples

```
## Not run:
# Create some dummy correlated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000L,
  ID = 2L,
  ZIP = 0L,
  AddDate = FALSE,
  Classification = TRUE,
  MultiClass = FALSE)

TestModel <- RemixAutoML::AutoH2oDRFClassifier(

    # Compute management
    MaxMem = "32G",
    NThreads = max(1L, parallel::detectCores() - 2L),
    IfSaveModel = "mojo",
    H2OShutdown = FALSE,

    # Metadata arguments:
    eval_metric = "auc",
    NumOfParDepPlots = 3L,

    # Data arguments:
    model_path = normalizePath("./"),
    metadata_path = NULL,
    ModelID = "FirstModel",
    ReturnModelObjects = TRUE,
    SaveModelObjects = FALSE,

    # Model evaluation:
    data,
    TrainOnFull = FALSE,
    ValidationData = NULL,
    TestData = NULL,
    TargetColumnName = "Adrian",
    FeatureColNames = names(data)[!names(data) %chin%
      c("IDcol_1", "IDcol_2","Adrian")],

    # Model args
```

```
    Trees = 50L,
    GridTune = FALSE,
    MaxModelsInGrid = 10L)

## End(Not run)
```

`AutoH2oDRFHurdleModel` *AutoH2oDRFHurdleModel is generalized hurdle modeling framework*

Description

AutoH2oDRFHurdleModel is generalized hurdle modeling framework

Usage

```
AutoH2oDRFHurdleModel(
  data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  Buckets = 0L,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  TransformNumericColumns = NULL,
  SplitRatios = c(0.7, 0.2, 0.1),
  ModelID = "ModelTest",
  Paths = NULL,
  MetaDataPaths = NULL,
  SaveModelObjects = TRUE,
  IfSaveModel = "mojo",
  MaxMem = "28G",
  NThreads = max(1L, parallel::detectCores() - 2L),
  Trees = 1000L,
  GridTune = TRUE,
  MaxModelsInGrid = 1L,
  NumOfParDepPlots = 10L,
  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.
`TrainOnFull`	Set to TRUE to train on full data
`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)

TransformNumericColumns
: Transform numeric column inside the AutoCatBoostRegression() function

SplitRatios
: Supply vector of partition ratios. For example, c(0.70,0.20,0,10).

ModelID
: Define a character name for your models

Paths
: The path to your folder where you want your model information saved

MetaDataPaths
: A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to Paths.

SaveModelObjects
: Set to TRUE to save the model objects to file in the folders listed in Paths

IfSaveModel
: Save as "mojo" or "standard"

MaxMem
: Set the maximum memory your system can provide

NThreads
: Set the number of threads you want to dedicate to the model building

Trees
: Default 1000

GridTune
: Set to TRUE if you want to grid tune the models

MaxModelsInGrid
: 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 AutoXGBoostRegression() 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 the grid used

See Also

Other Supervised Learning - Compound: AutoCatBoostHurdleModel(), AutoCatBoostSizeFreqDist(), AutoH2oGBMHurdleModel(), AutoH2oGBMSizeFreqDist(), AutoXGBoostHurdleModel()

Examples

```
## Not run:
Output <- AutoH2oDRFHurdleModel(
  data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  Buckets = 1L,
  TargetColumnName = "Target_Variable",
  FeatureColNames = 4:ncol(data),
  TransformNumericColumns = NULL,
  SplitRatios = c(0.7, 0.2, 0.1),
```

```
  NThreads = max(1L, parallel::detectCores()-2L),
  ModelID = "ModelID",
  Paths = NULL,
  MetaDataPaths = NULL,
  SaveModelObjects = TRUE,
  IfSaveModel = "mojo",
  MaxMem = "28G",
  NThreads = max(1L, parallel::detectCores()-2L),
  Trees = 1000L,
  GridTune = FALSE,
  MaxModelsInGrid = 1L,
  NumOfParDepPlots = 10L,
  PassInGrid = NULL)

## End(Not run)
```

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,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  eval_metric = "logloss",
  Trees = 50,
  GridTune = FALSE,
  MaxMem = "32G",
  NThreads = max(1, parallel::detectCores() - 2),
  MaxModelsInGrid = 2,
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = FALSE,
  HurdleModel = FALSE
)
```

Arguments

`data`	This is your data set for training and testing your model
`TrainOnFull`	Set to TRUE to train on full data
`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"
`NThreads`	Set the number of threads you want to dedicate to the model building
`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
`metadata_path`	A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to model_path.
`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
`H2OShutdown`	Set to TRUE to have H2O shutdown after running this function
`HurdleModel`	Leave set to FALSE

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 Automated Supervised Learning - Multiclass Classification: AutoCatBoostMultiClass(), AutoH2oGAMMultiClass(), AutoH2oGBMMultiClass(), AutoH2oGLMMultiClass(), AutoH2oMLMultiClass(), AutoXGBoostMultiClass()

Examples

```
## Not run:
# Create some dummy correlated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000L,
  ID = 2L,
  ZIP = 0L,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = TRUE)

# Run function
TestModel <- RemixAutoML::AutoH2oDRFMultiClass(
   data,
   TrainOnFull = FALSE,
   ValidationData = NULL,
   TestData = NULL,
   TargetColumnName = "Adrian",
   FeatureColNames = names(data)[!names(data) %chin%
     c("IDcol_1", "IDcol_2","Adrian")],
   eval_metric = "logloss",
   Trees = 50,
   GridTune = FALSE,
   MaxMem = "32G",
   NThreads = max(1, parallel::detectCores()-2),
   MaxModelsInGrid = 10,
   model_path = normalizePath("./"),
   metadata_path = file.path(normalizePath("./"),
     "MetaData"),
   ModelID = "FirstModel",
   ReturnModelObjects = TRUE,
   SaveModelObjects = FALSE,
   IfSaveModel = "mojo",
   H2OShutdown = FALSE,
   HurdleModel = FALSE)

## End(Not run)
```

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,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  TransformNumericColumns = NULL,
 Methods = c("BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson"),
  eval_metric = "RMSE",
  Trees = 50,
  GridTune = FALSE,
  MaxMem = "32G",
  NThreads = max(1, parallel::detectCores() - 2),
  MaxModelsInGrid = 2,
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  NumOfParDepPlots = 3,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = TRUE,
  HurdleModel = FALSE
)
```

Arguments

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

`TrainOnFull` Set to TRUE to train on full data

`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)

`TransformNumericColumns`
Set to NULL to do nothing; otherwise supply the column names of numeric variables you want transformed

`Methods` Choose from "BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson". Function will determine if one cannot be used because of the underlying data.

`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"

`NThreads` Set the number of threads you want to dedicate to the model building

`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

`metadata_path` A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to model_path.

`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

`H2OShutdown` For use in other functions.

`HurdleModel` Leave it set to FALSE

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, GridList, and Transformation metadata

Author(s)

Adrian Antico

See Also

Other Automated Supervised Learning - Regression: `AutoCatBoostRegression()`, `AutoH2oGAMRegression()`, `AutoH2oGBMRegression()`, `AutoH2oGLMRegression()`, `AutoH2oMLRegression()`, `AutoXGBoostRegression()`

Examples

```
## Not run:
# Create some dummy correlated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000,
  ID = 2,
  ZIP = 0,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run function
TestModel <- RemixAutoML::AutoH2oDRFRegression(

    # Compute management
    MaxMem = "32G",
    NThreads = max(1, parallel::detectCores()-2),
    H2OShutdown = TRUE,
    IfSaveModel = "mojo",

    # Model evaluation:
    eval_metric = "RMSE",
    NumOfParDepPlots = 3,

    # Metadata arguments:
    model_path = normalizePath("./"),
    metadata_path = NULL,
    ModelID = "FirstModel",
    ReturnModelObjects = TRUE,
    SaveModelObjects = FALSE,

    # Data arguments:
    TrainOnFull = FALSE,
    ValidationData = NULL,
    TestData = NULL,
    TargetColumnName = "Adrian",
    FeatureColNames = names(data)[!names(data) %chin%
      c("IDcol_1", "IDcol_2","Adrian")],
    TransformNumericColumns = NULL,
    Methods = c("BoxCox", "Asinh", "Asin", "Log",
      "LogPlus1", "Logit", "YeoJohnson"),

    # Model args
    Trees = 50,
    GridTune = FALSE,
    MaxModelsInGrid = 10)

## End(Not run)
```

`AutoH2oGAMClassifier` *AutoH2oGAMClassifier is an automated H2O modeling framework with grid-tuning and model evaluation*

Description

AutoH2oGAMClassifier 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

```
AutoH2oGAMClassifier(
  data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  GamColNames = NULL,
  Distribution = "binomial",
  link = "logit",
  eval_metric = "auc",
  GridTune = FALSE,
  MaxMem = "32G",
  NThreads = max(1, parallel::detectCores() - 2),
  MaxModelsInGrid = 2,
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  NumOfParDepPlots = 3,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = FALSE,
  HurdleModel = FALSE
)
```

Arguments

`data`	This is your data set for training and testing your model
`TrainOnFull`	Set to TRUE to train on full data
`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)

GamColNames
: GAM column names. Up to 9 features

Distribution
: "binomial", "quasibinomial"

link
: identity, logit, log, inverse, tweedie

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

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"

NThreads
: Set the number of threads you want to dedicate to the model building

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

metadata_path
: A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to model_path.

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

H2OShutdown
: Set to TRUE to shutdown H2O after running the function

HurdleModel
: Set to FALSE

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 Automated Supervised Learning - Binary Classification: AutoCatBoostClassifier(), AutoH2oDRFClassifier(), AutoH2oGBMClassifier(), AutoH2oGLMClassifier(), AutoH2oMLClassifier(), AutoXGBoostClassifier()

Examples

```
# Create some dummy correlated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000,
  ID = 2,
  ZIP = 0,
  AddDate = FALSE,
  Classification = TRUE,
  MultiClass = FALSE)

# Define GAM Columns to use - up to 9 are allowed
GamCols <- names(which(unlist(lapply(data, is.numeric))))
GamCols <- GamCols[!GamCols %in% c("Adrian","IDcol_1","IDcol_2")]
GamCols <- GamCols[1L:(min(9L,length(GamCols)))]

# Run function
TestModel <- RemixAutoML::AutoH2oGAMClassifier(

  # Compute management
  MaxMem = "32G",
  NThreads = max(1, parallel::detectCores()-2),
  H2OShutdown = TRUE,
  IfSaveModel = "mojo",

  # Model evaluation:
  eval_metric = "RMSE",
  NumOfParDepPlots = 3,

  # Metadata arguments:
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,

  # Data arguments:
  data = data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = "Adrian",
  FeatureColNames = names(data)[!names(data) %chin%
    c("IDcol_1", "IDcol_2","Adrian")],
  GamColNames = GamCols,

  # Model args
  GridTune = FALSE,
  MaxModelsInGrid = 10,
  Distribution = "binomial",
  link = "Family_Default",
  HurdleModel = FALSE)
```

`AutoH2oGAMMultiClass` *AutoH2oGAMMultiClass is an automated H2O modeling framework with grid-tuning and model evaluation*

Description

AutoH2oGAMMultiClass 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

```
AutoH2oGAMMultiClass(
  data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  GamColNames = NULL,
  eval_metric = "logloss",
  GridTune = FALSE,
  MaxMem = "32G",
  NThreads = max(1, parallel::detectCores() - 2),
  MaxModelsInGrid = 2,
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = FALSE,
  HurdleModel = FALSE
)
```

Arguments

`data`	This is your data set for training and testing your model
`TrainOnFull`	Set to TRUE to train on full data
`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)

GamColNames
: GAM column names. Up to 9 features

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

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"

NThreads
: Set the number of threads you want to dedicate to the model building

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

metadata_path
: A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to model_path.

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

H2OShutdown
: Set to TRUE to have H2O shutdown after running this function

HurdleModel
: Set to FALSE

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 Automated Supervised Learning - Multiclass Classification: AutoCatBoostMultiClass(), AutoH2oDRFMultiClass(), AutoH2oGBMMultiClass(), AutoH2oGLMMultiClass(), AutoH2oMLMultiClass(), AutoXGBoostMultiClass()

Examples

```
# Create some dummy correlated data with numeric and categorical features
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000L,
  ID = 2L,
  ZIP = 0L,
  AddDate = FALSE,
```

```
  Classification = FALSE,
  MultiClass = TRUE)

# Define GAM Columns to use - up to 9 are allowed
GamCols <- names(which(unlist(lapply(data, is.numeric))))
GamCols <- GamCols[!GamCols %in% c("Adrian","IDcol_1","IDcol_2")]
GamCols <- GamCols[1L:(min(9L,length(GamCols)))]

# Run function
TestModel <- RemixAutoML::AutoH2oGAMMultiClass(
   data,
   TrainOnFull = FALSE,
   ValidationData = NULL,
   TestData = NULL,
   TargetColumnName = "Adrian",
   FeatureColNames = names(data)[!names(data) %in%
     c("IDcol_1", "IDcol_2","Adrian")],
   GamColNames = GamCols,
   eval_metric = "logloss",
   GridTune = FALSE,
   MaxMem = "32G",
   NThreads = max(1, parallel::detectCores()-2),
   MaxModelsInGrid = 10,
   model_path = normalizePath("./"),
   metadata_path = NULL,
   ModelID = "FirstModel",
   ReturnModelObjects = TRUE,
   SaveModelObjects = FALSE,
   IfSaveModel = "mojo",
   H2OShutdown = FALSE,
   HurdleModel = FALSE)
```

AutoH2oGAMRegression *AutoH2oGAMRegression is an automated H2O modeling framework with grid-tuning and model evaluation*

Description

AutoH2oGAMRegression 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

```
AutoH2oGAMRegression(
  data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
```

```
  FeatureColNames = NULL,
  GamColNames = NULL,
  Distribution = "gaussian",
  link = "identity",
  TransformNumericColumns = NULL,
 Methods = c("BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson"),
  eval_metric = "RMSE",
  GridTune = FALSE,
  MaxMem = "32G",
  NThreads = max(1, parallel::detectCores() - 2),
  MaxModelsInGrid = 2,
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  NumOfParDepPlots = 3,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = TRUE,
  HurdleModel = FALSE
)
```

Arguments

`data`	This is your data set for training and testing your model
`TrainOnFull`	Set to TRUE to train on full data
`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)
`GamColNames`	GAM column names. Up to 9 features
`Distribution`	"binomial", "quasibinomial"
`link`	identity, logit, log, inverse, tweedie
`TransformNumericColumns`	Set to NULL to do nothing; otherwise supply the column names of numeric variables you want transformed
`Methods`	Choose from "BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson". Function will determine if one cannot be used because of the underlying data.
`eval_metric`	This is the metric used to identify best grid tuned model. Choose from "MSE", "RMSE", "MAE", "RMSLE"
`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"
`NThreads`	Set the number of threads you want to dedicate to the model building
`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
`metadata_path`	A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to model_path.
`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
`H2OShutdown`	For use in other functions.
`HurdleModel`	Set to FALSE

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, GridList, and Transformation metadata

Author(s)

Adrian Antico

See Also

Other Automated Supervised Learning - Regression: `AutoCatBoostRegression()`, `AutoH2oDRFRegression()`, `AutoH2oGBMRegression()`, `AutoH2oGLMRegression()`, `AutoH2oMLRegression()`, `AutoXGBoostRegression()`

Examples

```
# Create some dummy correlated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000,
  ID = 2,
  ZIP = 0,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)
```

```
# Define GAM Columns to use - up to 9 are allowed
GamCols <- names(which(unlist(lapply(data, is.numeric))))
GamCols <- GamCols[!GamCols %in% c("Adrian","IDcol_1","IDcol_2")]
GamCols <- GamCols[1L:(min(9L,length(GamCols)))]

# Run function
TestModel <- RemixAutoML::AutoH2oGAMRegression(

 # Compute management
 MaxMem = "32G",
 NThreads = max(1, parallel::detectCores()-2),
 H2OShutdown = TRUE,
 IfSaveModel = "mojo",

 # Model evaluation:
 eval_metric = "RMSE",
 NumOfParDepPlots = 3,

 # Metadata arguments:
 model_path = NULL,
 metadata_path = NULL,
 ModelID = "FirstModel",
 ReturnModelObjects = TRUE,
 SaveModelObjects = FALSE,

 # Data arguments:
 data = data,
 TrainOnFull = FALSE,
 ValidationData = NULL,
 TestData = NULL,
 TargetColumnName = "Adrian",
 FeatureColNames = names(data)[!names(data) %chin%
                                c("IDcol_1", "IDcol_2","Adrian")],
 GamColNames = GamCols,
 TransformNumericColumns = NULL,
 Methods = c("BoxCox", "Asinh", "Asin", "Log",
             "LogPlus1", "Logit", "YeoJohnson"),

 # Model args
 GridTune = FALSE,
 MaxModelsInGrid = 10,
 Distribution = "gaussian",
 link = "Family_Default")
```

`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,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  eval_metric = "auc",
  Trees = 50L,
  GridTune = FALSE,
  MaxMem = "32G",
  NThreads = max(1L, parallel::detectCores() - 2L),
  MaxModelsInGrid = 2L,
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  NumOfParDepPlots = 3L,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = FALSE,
  HurdleModel = FALSE
)
```

Arguments

`data`	This is your data set for training and testing your model
`TrainOnFull`	Set to TRUE to train on full data
`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"

NThreads — Set to the number of threads you want to use for running this function

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

metadata_path — A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to model_path.

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

H2OShutdown — Set to TRUE to shut down H2O after running the function

HurdleModel — Set to FALSE

Value

Saves to file and returned in list: VariableImportance.csv, Model, ValidationData.csv, Evalution-Plot.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 Automated Supervised Learning - Binary Classification: AutoCatBoostClassifier(), AutoH2oDRFClassifier(), AutoH2oGAMClassifier(), AutoH2oGLMClassifier(), AutoH2oMLClassifier(), AutoXGBoostClassifier()

Examples

```
## Not run:
# Create some dummy correlated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000L,
  ID = 2L,
  ZIP = 0L,
  AddDate = FALSE,
  Classification = TRUE,
  MultiClass = FALSE)

TestModel <- RemixAutoML::AutoH2oGBMClassifier(

    # Compute management
```

```
    MaxMem = "32G",
    NThreads = max(1, parallel::detectCores()-2),
    H2OShutdown = FALSE,
    IfSaveModel = "mojo",

    # Model evaluation:
    eval_metric = "auc",
    NumOfParDepPlots = 3L,

    # Metadata arguments:
    ModelID = "FirstModel",
    model_path = normalizePath("./"),
    metadata_path = file.path(normalizePath("./"),
      "MetaData"),
    ReturnModelObjects = TRUE,
    SaveModelObjects = FALSE,

    # Data arguments:
    data,
    TrainOnFull = FALSE,
    ValidationData = NULL,
    TestData = NULL,
    TargetColumnName = "Adrian",
    FeatureColNames = names(data)[!names(data) %chin%
      c("IDcol_1", "IDcol_2","Adrian")],

    # Model args
    Trees = 50,
    GridTune = FALSE,
    MaxModelsInGrid = 10L)

## End(Not run)
```

AutoH2oGBMFreqSizeScoring *AutoH2oGBMFreqSizeScoring is for scoring the models build with AutoH2oGBMSizeFreqDist()*

Description

AutoH2oGBMFreqSizeScoring is for scoring the models build with AutoH2oGBMSizeFreqDist(). It will return the predicted values for every quantile model for both distributions for 1 to the max forecast periods you provided to build the scoring data.

Usage

```
AutoH2oGBMFreqSizeScoring(
  ScoringData,
  TargetColumnNames = NULL,
  CountQuantiles = seq(0.1, 0.9, 0.1),
  SizeQuantiles = seq(0.1, 0.9, 0.1),
  ModelPath = NULL,
  ModelIDs = c("CountModel", "SizeModel"),
```

```
  JavaOptions = "-Xmx1g -XX:ReservedCodeCacheSize=256m",
  KeepFeatures = TRUE
)
```

Arguments

ScoringData	The scoring data returned from IntermittentDemandScoringDataGenerator()
TargetColumnNames	A character or numeric vector of the target names. E.g. c("Counts","TARGET_qty")
CountQuantiles	A numerical vector of the quantiles used in model building
SizeQuantiles	A numerical vector of the quantiles used in model building
ModelPath	The path file to where you models were saved
ModelIDs	The ID's used in model building
JavaOptions	For mojo scoring '-Xmx1g -XX:ReservedCodeCacheSize=256m',
KeepFeatures	Set to TRUE to return the features with the predicted values

Value

Returns a list of CountData scores, SizeData scores, along with count and size prediction column names

Author(s)

Adrian Antico

See Also

Other Automated Time Series: AutoBanditNNet(), AutoBanditSarima(), AutoCatBoostFreqSizeScoring(), AutoTBATS(), AutoTS()

Examples

```
## Not run:
FinalData <- AutoH2oGBMFreqSizeScoring(
  ScoringData,
  TargetColumnNames = c("Counts","TARGET_qty"),
  CountQuantiles = seq(0.10,0.90,0.10),
  SizeQuantiles = seq(0.10,0.90,0.10),
  ModelPath = getwd(),
  ModelIDs = c("CountModel","SizeModel"),
  JavaOptions = '-Xmx1g -XX:ReservedCodeCacheSize=256m',
  KeepFeatures = TRUE)

## End(Not run)
```

AutoH2oGBMHurdleModel *AutoH2oGBMHurdleModel is generalized hurdle modeling framework*

Description

AutoH2oGBMHurdleModel is generalized hurdle modeling framework

Usage

```
AutoH2oGBMHurdleModel(
  data,
  ValidationData = NULL,
  TestData = NULL,
  Buckets = 0L,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  TransformNumericColumns = NULL,
  Distribution = "gaussian",
  SplitRatios = c(0.7, 0.2, 0.1),
  ModelID = "ModelTest",
  Paths = NULL,
  MetaDataPaths = NULL,
  SaveModelObjects = TRUE,
  IfSaveModel = "mojo",
  MaxMem = "28G",
  NThreads = max(1L, parallel::detectCores() - 2L),
  Trees = 1000L,
  GridTune = TRUE,
  MaxModelsInGrid = 1L,
  NumOfParDepPlots = 10L,
  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)

TransformNumericColumns
: Transform numeric column inside the AutoCatBoostRegression() function

Distribution
: Set to the distribution of choice based on H2O regression documents.

SplitRatios
: Supply vector of partition ratios. For example, c(0.70,0.20,0,10).

ModelID
: Define a character name for your models

Paths
: The path to your folder where you want your model information saved

MetaDataPaths
: A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to Paths.

SaveModelObjects
: Set to TRUE to save the model objects to file in the folders listed in Paths

IfSaveModel
: Save as "mojo" or "standard"

MaxMem
: Set the maximum memory your system can provide

NThreads
: Set the number of threads you want to dedicate to the model building

Trees
: Default 1000

GridTune
: Set to TRUE if you want to grid tune the models

MaxModelsInGrid
: 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 AutoXGBoostRegression() 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 the grid used

See Also

Other Supervised Learning - Compound: AutoCatBoostHurdleModel(), AutoCatBoostSizeFreqDist(), AutoH2oDRFHurdleModel(), AutoH2oGBMSizeFreqDist(), AutoXGBoostHurdleModel()

Examples

```
Output <- RemixAutoML::AutoH2oGBMHurdleModel(
  data,
  ValidationData = NULL,
  TestData = NULL,
  Buckets = 1L,
  TargetColumnName = "Target_Variable",
  FeatureColNames = 4L:ncol(data),
  TransformNumericColumns = NULL,
  Distribution = "gaussian",
  SplitRatios = c(0.7, 0.2, 0.1),
  NThreads = max(1L, parallel::detectCores()-2L),
  ModelID = "ModelID",
  Paths = normalizePath("./"),
  MetaDataPaths = NULL,
  SaveModelObjects = TRUE,
  IfSaveModel = "mojo",
```

```
  Trees = 1000L,
  GridTune = FALSE,
  MaxModelsInGrid = 1L,
  NumOfParDepPlots = 10L,
  PassInGrid = NULL)
```

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,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  eval_metric = "logloss",
  Trees = 50,
  GridTune = FALSE,
  MaxMem = "32G",
  NThreads = max(1, parallel::detectCores() - 2),
  MaxModelsInGrid = 2,
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = FALSE,
  HurdleModel = FALSE
)
```

Arguments

data	This is your data set for training and testing your model
TrainOnFull	Set to TRUE to train on full data
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"

`NThreads` Set to the number of threads you want to use for running this function

`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

`metadata_path` A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to model_path.

`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

`H2OShutdown` Set to TRUE to shutdown H2O when done with function

`HurdleModel` Set to FALSE

Value

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

Author(s)

Adrian Antico

See Also

Other Automated Supervised Learning - Multiclass Classification: `AutoCatBoostMultiClass()`, `AutoH2oDRFMultiClass()`, `AutoH2oGAMMultiClass()`, `AutoH2oGLMMultiClass()`, `AutoH2oMLMultiClass()`, `AutoXGBoostMultiClass()`

Examples

```
# Create some dummy correlated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000,
  ID = 2,
  ZIP = 0,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = TRUE)

# Run function
TestModel <- RemixAutoML::AutoH2oGBMMultiClass(
   data,
   TrainOnFull = FALSE,
   ValidationData = NULL,
   TestData = NULL,
   TargetColumnName = "Adrian",
   FeatureColNames = names(data)[!names(data) %chin%
     c("IDcol_1", "IDcol_2","Adrian")],
   eval_metric = "logloss",
   Trees = 50,
   GridTune = FALSE,
   MaxMem = "32G",
   NThreads = max(1, parallel::detectCores()-2),
   MaxModelsInGrid = 10,
   model_path = normalizePath("./"),
   metadata_path = file.path(normalizePath("./"),
     "MetaData"),
   ModelID = "FirstModel",
   ReturnModelObjects = TRUE,
   SaveModelObjects = FALSE,
   IfSaveModel = "mojo",
   H2OShutdown = FALSE,
   HurdleModel = FALSE)
```

`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,
  TrainOnFull = FALSE,
  ValidationData,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  TransformNumericColumns = NULL,
  Methods = c("BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson"),
  Alpha = NULL,
  Distribution = "poisson",
  eval_metric = "RMSE",
  Trees = 50,
  GridTune = FALSE,
  MaxMem = "32G",
  NThreads = max(1, parallel::detectCores() - 2),
  MaxModelsInGrid = 2,
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  NumOfParDepPlots = 3,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = TRUE,
  HurdleModel = FALSE
)
```

Arguments

- `data` This is your data set for training and testing your model
- `TrainOnFull` Set to TRUE to train on full data
- `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)
- `TransformNumericColumns` Set to NULL to do nothing; otherwise supply the column names of numeric variables you want transformed
- `Methods` Choose from "BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson". Function will determine if one cannot be used because of the underlying data.
- `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"

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"

NThreads Set to the mamimum amount of threads you want to use for this function

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

metadata_path A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to model_path.

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

H2OShutdown Set to FALSE to keep H2O running after you build your model

HurdleModel Set to FALSE

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, GridList, and metadata

Author(s)

Adrian Antico

See Also

Other Automated Supervised Learning - Regression: AutoCatBoostRegression(), AutoH2oDRFRegression(), AutoH2oGAMRegression(), AutoH2oGLMRegression(), AutoH2oMLRegression(), AutoXGBoostRegression()

Examples

```
# Create some dummy correlated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000,
  ID = 2,
  ZIP = 0,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run function
TestModel <- RemixAutoML::AutoH2oGBMRegression(

    # Compute management
    MaxMem = "32G",
    NThreads = max(1, parallel::detectCores()-2),
    H2OShutdown = TRUE,
    IfSaveModel = "mojo",
    Alpha = NULL,
    Distribution = "poisson",

    # Model evaluation:
    eval_metric = "RMSE",
    NumOfParDepPlots = 3,

    # Metadata arguments:
    model_path = NULL,
    metadata_path = NULL,
    ModelID = "FirstModel",
    ReturnModelObjects = TRUE,
    SaveModelObjects = FALSE,

    # Data arguments:
    TrainOnFull = FALSE,
    ValidationData = NULL,
    TestData = NULL,
    TargetColumnName = "Adrian",
    FeatureColNames = names(data)[!names(data) %chin%
      c("IDcol_1", "IDcol_2","Adrian")],
    TransformNumericColumns = NULL,
    Methods = c("BoxCox", "Asinh", "Asin", "Log",
                "LogPlus1","Logit","YeoJohnson"),

    # Model args
    Trees = 50,
    GridTune = FALSE,
    MaxModelsInGrid = 10)
```

`AutoH2oGBMSizeFreqDist` *AutoH2oGBMSizeFreqDist for building size and frequency distributions via quantile regressions*

Description

AutoH2oGBMSizeFreqDist for building size and frequency distributions via quantile regressions. Size (or severity) and frequency (or count) quantile regressions are build. Use this with the ID_SingleLevelGibbsSampler function to simulate the joint distribution.

Usage

```
AutoH2oGBMSizeFreqDist(
  CountData = NULL,
  SizeData = NULL,
  CountQuantiles = seq(0.1, 0.9, 0.1),
  SizeQuantiles = seq(0.1, 0.9, 0.1),
  AutoTransform = TRUE,
  DataPartitionRatios = c(0.75, 0.2, 0.05),
  StratifyColumnName = NULL,
  StratifyTargets = FALSE,
  NTrees = 1500,
  MaxMem = "28G",
  NThreads = max(1, parallel::detectCores() - 2),
  EvalMetric = "Quantile",
  GridTune = FALSE,
  CountTargetColumnName = NULL,
  SizeTargetColumnName = NULL,
  CountFeatureColNames = NULL,
  SizeFeatureColNames = NULL,
  ModelIDs = c("CountModel", "SizeModel"),
  MaxModelsGrid = 5,
  ModelPath = NULL,
  MetaDataPath = NULL,
  NumOfParDepPlots = 0
)
```

Arguments

`CountData` This is your CountData generated from the IntermittentDemandBootStrapper() function

`SizeData` This is your SizeData generated from the IntermittentDemandBootStrapper() function

`CountQuantiles` The default are deciles, i.e. seq(0.10,0.90,0.10). More granularity the better, but it will take longer to run.

`SizeQuantiles` The default are deciles, i.e. seq(0.10,0.90,0.10). More granularity the better, but it will take longer to run.

`AutoTransform` Set to FALSE not to have the your target variables automatically transformed for the best normalization.

`DataPartitionRatios` The default is c(0.75,0.20,0.05). With CatBoost, you should allocate a decent amount to the validation data (second input). Three inputs are required.

`StratifyColumnName` You can specify grouping columns to stratify by

`StratifyTargets`
: Set to TRUE to stratify by the target variables to ensure the a more even allocation for potentially highly skewed data

`NTrees`
: Default is 1500. If the best model utilizes all trees, you should consider increasing the argument.

`MaxMem`
: The max memory allocation. E.g. "28G"

`NThreads`
: The max threads to use. E.g. 4

`EvalMetric`
: Set to "Quantile". Alternative quantile methods may become available in the future.

`GridTune`
: The default is set to FALSE. If you set to TRUE, make sure to specify MaxModelsGrid to a number greater than 1.

`CountTargetColumnName`
: Column names or column numbers

`SizeTargetColumnName`
: Column names or column numbers

`CountFeatureColNames`
: Column names or column numbers

`SizeFeatureColNames`
: Column names or column numbers

`ModelIDs`
: A two element character vector. E.g. c("CountModel","SizeModel")

`MaxModelsGrid`
: Set to a number greater than 1 if GridTune is set to TRUE

`ModelPath`
: This path file is where all your models will be stored. If you leave MetaDataPath NULL, the evaluation metadata will also be stored here. If you leave this NULL, the function will not run.

`MetaDataPath`
: A separate path to store the model metadata for evaluation.

`NumOfParDepPlots`
: Set to a number greater than or equal to 1 to see the relationships between your features and targets.

Value

This function does not return anything. It can only store your models and model evaluation metadata to file.

Author(s)

Adrian Antico

See Also

Other Supervised Learning - Compound: `AutoCatBoostHurdleModel()`, `AutoCatBoostSizeFreqDist()`, `AutoH2oDRFHurdleModel()`, `AutoH2oGBMHurdleModel()`, `AutoXGBoostHurdleModel()`

Examples

```
AutoH2oGBMSizeFreqDist(
  CountData = NULL,
  SizeData = NULL,
  CountQuantiles = seq(0.10,0.90,0.10),
  SizeQuantiles = seq(0.10,0.90,0.10),
  AutoTransform = TRUE,
```

```
    DataPartitionRatios = c(0.75,0.20,0.05),
    StratifyColumnName = NULL,
    StratifyTargets = FALSE,
    NTrees = 1500,
    MaxMem = "28G",
    NThreads = max(1, parallel::detectCores()-2),
    EvalMetric = "Quantile",
    GridTune = FALSE,
    CountTargetColumnName = NULL,
    SizeTargetColumnName = NULL,
    CountFeatureColNames = NULL,
    SizeFeatureColNames = NULL,
    ModelIDs = c("CountModel","SizeModel"),
    MaxModelsGrid = 5,
    ModelPath = NULL,
    MetaDataPath = NULL,
    NumOfParDepPlots = 0)
```

`AutoH2oGLMClassifier` *AutoH2oGLMClassifier is an automated H2O modeling framework with grid-tuning and model evaluation*

Description

AutoH2oGLMClassifier 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

```
AutoH2oGLMClassifier(
  data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  Distribution = "binomial",
  link = "logit",
  eval_metric = "auc",
  GridTune = FALSE,
  MaxMem = "32G",
  NThreads = max(1, parallel::detectCores() - 2),
  MaxModelsInGrid = 2,
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
```

```
  NumOfParDepPlots = 3,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = FALSE,
  HurdleModel = FALSE
)
```

Arguments

`data`	This is your data set for training and testing your model
`TrainOnFull`	Set to TRUE to train on full data
`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)
`Distribution`	"binomial", "quasibinomial"
`link`	identity, logit, log, inverse, tweedie
`eval_metric`	This is the metric used to identify best grid tuned model. Choose from "AUC" or "logloss"
`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"
`NThreads`	Set the number of threads you want to dedicate to the model building
`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
`metadata_path`	A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to model_path.
`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
`H2OShutdown`	Set to TRUE to shutdown H2O after running the function
`HurdleModel`	Set to FALSE

Value

Saves to file and returned in list: VariableImportance.csv, Model, ValidationData.csv, Evalution-Plot.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 Automated Supervised Learning - Binary Classification: AutoCatBoostClassifier(), AutoH2oDRFClassifier(), AutoH2oGAMClassifier(), AutoH2oGBMClassifier(), AutoH2oMLClassifier(), AutoXGBoostClassifier()

Examples

```
# Create some dummy correlated data with numeric and categorical features
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000L,
  ID = 2L,
  ZIP = 0L,
  AddDate = FALSE,
  Classification = TRUE,
  MultiClass = FALSE)

# Run function
TestModel <- RemixAutoML::AutoH2oGLMClassifier(
   data,
   TrainOnFull = FALSE,
   ValidationData = NULL,
   TestData = NULL,
   TargetColumnName = "Adrian",
   FeatureColNames = names(data)[!names(data) %chin% c("IDcol_1", "IDcol_2","Adrian")],
   Distribution = "binomial",
   link = "logit",
   eval_metric = "auc",
   GridTune = FALSE,
   MaxMem = "32G",
   NThreads = max(1, parallel::detectCores()-2),
   MaxModelsInGrid = 10,
   model_path = NULL,
   metadata_path = NULL,
   ModelID = "FirstModel",
   NumOfParDepPlots = 3,
   ReturnModelObjects = TRUE,
   SaveModelObjects = FALSE,
   IfSaveModel = "mojo",
   H2OShutdown = FALSE,
   HurdleModel = FALSE)
```

`AutoH2oGLMMultiClass` *AutoH2oGLMMultiClass is an automated H2O modeling framework with grid-tuning and model evaluation*

Description

AutoH2oGLMMultiClass 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

```
AutoH2oGLMMultiClass(
  data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  eval_metric = "logloss",
  GridTune = FALSE,
  MaxMem = "32G",
  NThreads = max(1, parallel::detectCores() - 2),
  MaxModelsInGrid = 2,
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = FALSE,
  HurdleModel = FALSE
)
```

Arguments

`data`	This is your data set for training and testing your model
`TrainOnFull`	Set to TRUE to train on full data
`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"

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"

NThreads
: Set the number of threads you want to dedicate to the model building

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

metadata_path
: A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to model_path.

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

H2OShutdown
: Set to TRUE to have H2O shutdown after running this function

HurdleModel
: Set to FALSE

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 Automated Supervised Learning - Multiclass Classification: AutoCatBoostMultiClass(), AutoH2oDRFMultiClass(), AutoH2oGAMMultiClass(), AutoH2oGBMMultiClass(), AutoH2oMLMultiClass(), AutoXGBoostMultiClass()

Examples

```
# Create some dummy correlated data with numeric and categorical features
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000L,
  ID = 2L,
  ZIP = 0L,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = TRUE)
```

```
# Run function
TestModel <- RemixAutoML::AutoH2oGLMMultiClass(
   data,
   TrainOnFull = FALSE,
   ValidationData = NULL,
   TestData = NULL,
   TargetColumnName = "Adrian",
   FeatureColNames = names(data)[!names(data) %chin% c("IDcol_1", "IDcol_2","Adrian")],
   eval_metric = "logloss",
   GridTune = FALSE,
   MaxMem = "32G",
   NThreads = max(1, parallel::detectCores()-2),
   MaxModelsInGrid = 10,
   model_path = normalizePath("./"),
   metadata_path = file.path(normalizePath("./"), "MetaData"),
   ModelID = "FirstModel",
   ReturnModelObjects = TRUE,
   SaveModelObjects = FALSE,
   IfSaveModel = "mojo",
   H2OShutdown = FALSE,
   HurdleModel = FALSE)
```

`AutoH2oGLMRegression` *AutoH2oGLMRegression is an automated H2O modeling framework with grid-tuning and model evaluation*

Description

AutoH2oGLMRegression 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

```
AutoH2oGLMRegression(
  data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  Distribution = "gaussian",
  link = "identity",
  TransformNumericColumns = NULL,
  Methods = c("BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson"),
  eval_metric = "RMSE",
  GridTune = FALSE,
  MaxMem = "32G",
```

```
  NThreads = max(1, parallel::detectCores() - 2),
  MaxModelsInGrid = 2,
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  NumOfParDepPlots = 3,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = TRUE,
  HurdleModel = FALSE
)
```

Arguments

- `data` This is your data set for training and testing your model
- `TrainOnFull` Set to TRUE to train on full data
- `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)
- `Distribution` "binomial", "quasibinomial"
- `link` identity, logit, log, inverse, tweedie
- `TransformNumericColumns` Set to NULL to do nothing; otherwise supply the column names of numeric variables you want transformed
- `Methods` Choose from "BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson". Function will determine if one cannot be used because of the underlying data.
- `eval_metric` This is the metric used to identify best grid tuned model. Choose from "MSE", "RMSE", "MAE", "RMSLE"
- `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"
- `NThreads` Set the number of threads you want to dedicate to the model building
- `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
- `metadata_path` A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to model_path.

`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

`H2OShutdown` For use in other functions.

`HurdleModel` Set to FALSE

Value

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

Author(s)

Adrian Antico

See Also

Other Automated Supervised Learning - Regression: AutoCatBoostRegression(), AutoH2oDRFRegression(), AutoH2oGAMRegression(), AutoH2oGBMRegression(), AutoH2oMLRegression(), AutoXGBoostRegression()

Examples

```
# Create some dummy correlated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000,
  ID = 2,
  ZIP = 0,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run function
TestModel <- RemixAutoML::AutoH2oGLMRegression(

    # Compute management
    MaxMem = "32G",
    NThreads = max(1, parallel::detectCores()-2),
    H2OShutdown = TRUE,
    IfSaveModel = "mojo",

    # Model evaluation:
    eval_metric = "RMSE",
    NumOfParDepPlots = 3,
```

```
# Metadata arguments:
model_path = NULL,
metadata_path = NULL,
ModelID = "FirstModel",
ReturnModelObjects = TRUE,
SaveModelObjects = FALSE,

# Data arguments:
TrainOnFull = FALSE,
ValidationData = NULL,
TestData = NULL,
TargetColumnName = "Adrian",
FeatureColNames = names(data)[!names(data) %chin%
  c("IDcol_1", "IDcol_2","Adrian")],
TransformNumericColumns = NULL,
Methods = c("BoxCox", "Asinh", "Asin", "Log",
            "LogPlus1", "Logit", "YeoJohnson"),

# Model args
GridTune = FALSE,
MaxModelsInGrid = 10,
Distribution = "gaussian",
link = "identity")
```

AutoH2oMLClassifier *AutoH2oMLClassifier is an automated H2O modeling framework with grid-tuning and model evaluation*

Description

AutoH2oMLClassifier 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

```
AutoH2oMLClassifier(
  data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  ExcludeAlgos = NULL,
  eval_metric = "auc",
  Trees = 50,
  MaxMem = "32G",
```

```
  NThreads = max(1, parallel::detectCores() - 2),
  MaxModelsInGrid = 2,
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  NumOfParDepPlots = 3,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = FALSE,
  HurdleModel = FALSE
)
```

Arguments

- `data` This is your data set for training and testing your model
- `TrainOnFull` Set to TRUE to train on full data
- `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)
- `ExcludeAlgos` "DRF","GLM","XGBoost","GBM","DeepLearning" and "Stacke-dEnsemble"
- `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
- `MaxMem` Set the maximum amount of memory you'd like to dedicate to the model run. E.g. "32G"
- `NThreads` Set the number of threads you want to dedicate to the model building
- `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
- `metadata_path` A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to model_path.
- `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
H2OShutdown	Set to TRUE to shutdown H2O after running the function
HurdleModel	Set to FALSE

Value

Saves to file and returned in list: VariableImportance.csv, Model, ValidationData.csv, Evalution-Plot.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 Automated Supervised Learning - Binary Classification: `AutoCatBoostClassifier()`, `AutoH2oDRFClassifier()`, `AutoH2oGAMClassifier()`, `AutoH2oGBMClassifier()`, `AutoH2oGLMClassifier()`, `AutoXGBoostClassifier()`

Examples

```
# Create some dummy correlated data with numeric and categorical features
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000L,
  ID = 2L,
  ZIP = 0L,
  AddDate = FALSE,
  Classification = TRUE,
  MultiClass = FALSE)

TestModel <- RemixAutoML::AutoH2oMLClassifier(
   data,
   TrainOnFull = FALSE,
   ValidationData = NULL,
   TestData = NULL,
   TargetColumnName = "Adrian",
   FeatureColNames = names(data)[!names(data) %chin%
     c("IDcol_1", "IDcol_2","Adrian")],
   ExcludeAlgos = NULL,
   eval_metric = "auc",
   Trees = 50,
   MaxMem = "32G",
   NThreads = max(1, parallel::detectCores()-2),
   MaxModelsInGrid = 10,
   model_path = normalizePath("./"),
   metadata_path = file.path(normalizePath("./"), "MetaData"),
   ModelID = "FirstModel",
   NumOfParDepPlots = 3,
   ReturnModelObjects = TRUE,
   SaveModelObjects = FALSE,
   IfSaveModel = "mojo",
   H2OShutdown = FALSE,
   HurdleModel = FALSE)
```

AutoH2oMLMultiClass *AutoH2oMLMultiClass 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

```
AutoH2oMLMultiClass(
  data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  ExcludeAlgos = NULL,
  eval_metric = "logloss",
  Trees = 50,
  MaxMem = "32G",
  NThreads = max(1, parallel::detectCores() - 2),
  MaxModelsInGrid = 2,
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = FALSE,
  HurdleModel = FALSE
)
```

Arguments

`data`	This is your data set for training and testing your model
`TrainOnFull`	Set to TRUE to train on full data
`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)

ExcludeAlgos
: "DRF","GLM","XGBoost","GBM","DeepLearning" and "Stacke-dEnsemble"

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

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

NThreads
: Set the number of threads you want to dedicate to the model building

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

metadata_path
: A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to model_path.

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

H2OShutdown
: Set to TRUE to have H2O shutdown after running this function

HurdleModel
: Set to FALSE

Value

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

Author(s)

Adrian Antico

See Also

Other Automated Supervised Learning - Multiclass Classification: AutoCatBoostMultiClass(), AutoH2oDRFMultiClass(), AutoH2oGAMMultiClass(), AutoH2oGBMMultiClass(), AutoH2oGLMMultiClass(), AutoXGBoostMultiClass()

Examples

```
# Create some dummy correlated data with numeric and categorical features
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000,
  ID = 2,
  ZIP = 0,
  AddDate = FALSE,
  Classification = FALSE,
```

```
  MultiClass = TRUE)

# Run function
TestModel <- RemixAutoML::AutoH2oMLMultiClass(
   data,
   TrainOnFull = FALSE,
   ValidationData = NULL,
   TestData = NULL,
   TargetColumnName = "Adrian",
   FeatureColNames = names(data)[!names(data) %chin% c("IDcol_1", "IDcol_2","Adrian")],
   ExcludeAlgos = NULL,
   eval_metric = "logloss",
   Trees = 50,
   MaxMem = "32G",
   NThreads = max(1, parallel::detectCores()-2),
   MaxModelsInGrid = 10,
   model_path = normalizePath("./"),
   metadata_path = file.path(normalizePath("./"), "MetaData"),
   ModelID = "FirstModel",
   ReturnModelObjects = TRUE,
   SaveModelObjects = FALSE,
   IfSaveModel = "mojo",
   H2OShutdown = FALSE,
   HurdleModel = FALSE)
```

AutoH2oMLRegression *AutoH2oMLRegression is an automated H2O modeling framework with grid-tuning and model evaluation*

Description

AutoH2oMLRegression 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

```
AutoH2oMLRegression(
  data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  ExcludeAlgos = NULL,
  TransformNumericColumns = NULL,
 Methods = c("BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson"),
  eval_metric = "RMSE",
```

```
  Trees = 50,
  MaxMem = "32G",
  NThreads = max(1, parallel::detectCores() - 2),
  MaxModelsInGrid = 2,
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  NumOfParDepPlots = 3,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = TRUE,
  HurdleModel = FALSE
)
```

Arguments

`data`	This is your data set for training and testing your model
`TrainOnFull`	Set to TRUE to train on full data
`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)
`ExcludeAlgos`	"DRF","GLM","XGBoost","GBM","DeepLearning" and "Stacke-dEnsemble"
`TransformNumericColumns`	Set to NULL to do nothing; otherwise supply the column names of numeric variables you want transformed
`Methods`	Choose from "BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson". Function will determine if one cannot be used because of the underlying data.
`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
`MaxMem`	Set the maximum amount of memory you'd like to dedicate to the model run. E.g. "32G"
`NThreads`	Set the number of threads you want to dedicate to the model building
`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
`metadata_path`	A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to model_path.

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

H2OShutdown For use in other functions.

HurdleModel Set to FALSE

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, GridList, and Transformation metadata

Author(s)

Adrian Antico

See Also

Other Automated Supervised Learning - Regression: AutoCatBoostRegression(), AutoH2oDRFRegression(), AutoH2oGAMRegression(), AutoH2oGBMRegression(), AutoH2oGLMRegression(), AutoXGBoostRegression()

Examples

```
# Create some dummy correlated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000,
  ID = 2,
  ZIP = 0,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run function
TestModel <- RemixAutoML::AutoH2oMLRegression(

    # Compute management
    MaxMem = "32G",
    NThreads = max(1, parallel::detectCores()-2),
    H2OShutdown = TRUE,
    IfSaveModel = "mojo",

    # Model evaluation:
    #   'eval_metric' is the measure catboost uses when
    #      evaluting on holdout data during its bandit style
```

```
#      process
#   'NumOfParDepPlots' Number of partial dependence
#      calibration plots generated.
#     A value of 3 will return plots for the top 3 variables
#      based on variable importance
#     Won't be returned if GrowPolicy is either
#      "Depthwise" or "Lossguide" is used
#     Can run the RemixAutoML::ParDepCalPlots() with
#       the outputted ValidationData
eval_metric = "RMSE",
NumOfParDepPlots = 3,

# Metadata arguments:
#   'ModelID' is used to create part of the file names
#     generated when saving to file'
#   'model_path' is where the minimal model objects
#     for scoring will be stored
#      'ModelID' will be the name of the saved model object
#   'metadata_path' is where model evaluation and model
#      interpretation files are saved
#      objects saved to model_path if metadata_path is null
#      Saved objects include:
#         'ModelID_ValidationData.csv' is the supplied or
#            generated TestData with predicted values
#         'ModelID_VariableImportance.csv' is the variable
#            importance.
#            This won't be saved to file if GrowPolicy is either
#            "Depthwise" or "Lossguide" was used
#         'ModelID_ExperimentGrid.csv' if GridTune = TRUE.
#            Results of all model builds including parameter
#            settings, bandit probs, and grid IDs
#         'ModelID_EvaluationMetrics.csv' which contains MSE,
#          MAE, MAPE, R2
model_path = NULL,
metadata_path = NULL,
ModelID = "FirstModel",
ReturnModelObjects = TRUE,
SaveModelObjects = FALSE,

# Data arguments:
#   'TrainOnFull' is to train a model with 100
#      percent of your data.
#     That means no holdout data will be used for evaluation
#   If ValidationData and TestData are NULL and TrainOnFull
#      is FALSE then data will be split 70 20 10
#   'PrimaryDateColumn' is a date column in data that is
#      meaningful when sorted.
#     CatBoost categorical treatment is enhanced when supplied
#   'IDcols' are columns in your data that you don't use for
#      modeling but get returned with ValidationData
#   'TransformNumericColumns' is for transforming your target
#     variable. Just supply the name of it
TrainOnFull = FALSE,
ValidationData = NULL,
TestData = NULL,
TargetColumnName = "Adrian",
FeatureColNames = names(data)[!names(data) %chin%
```

```
      c("IDcol_1", "IDcol_2","Adrian")],
    TransformNumericColumns = NULL,
    Methods = c("BoxCox", "Asinh", "Asin", "Log",
      "LogPlus1", "Logit", "YeoJohnson"),

    # Model args
    ExcludeAlgos = NULL,
    Trees = 50,
    MaxModelsInGrid = 10)
```

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,
  ModelObject = NULL,
  ModelType = "mojo",
  H2OShutdown = TRUE,
  MaxMem = "28G",
  NThreads = max(1, parallel::detectCores() - 2),
  JavaOptions = "-Xmx1g -XX:ReservedCodeCacheSize=256m",
  ModelPath = NULL,
  ModelID = NULL,
  ReturnFeatures = TRUE,
  TransformNumeric = FALSE,
  BackTransNumeric = FALSE,
  TargetColumnName = NULL,
  TransformationObject = NULL,
  TransID = NULL,
  TransPath = NULL,
  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.
ModelObject	Supply a model object from AutoH2oDRF__()

- `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"
- `NThreads` Default set to max(1, parallel::detectCores()-2)
- `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.
- `TransformNumeric` Set to TRUE if you have features that were transformed automatically from an Auto__Regression() model AND you haven't already transformed them.
- `BackTransNumeric` Set to TRUE to generate back-transformed predicted values. Also, if you return features, those will also be back-transformed.
- `TargetColumnName` Input your target column name used in training if you are utilizing the transformation service
- `TransformationObject` Set to NULL if you didn't use transformations or if you want the function to pull from the file output from the Auto__Regression() function. You can also supply the transformation data.table object with the transformation details versus having it pulled from file.
- `TransID` Set to the ID used for saving the transformation data.table object or set it to the ModelID if you are pulling from file from a build with Auto__Regression().
- `TransPath` Set the path file to the folder where your transformation data.table detail object is stored. If you used the Auto__Regression() to build, set it to the same path as ModelPath.
- `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.

Author(s)

Adrian Antico

See Also

Other Automated Model Scoring: AutoCatBoostScoring(), AutoH2OModeler(), AutoHurdleScoring(), AutoXGBoostScoring(), IntermittentDemandScoringDataGenerator()

Examples

```
## Not run:
Preds <- AutoH2OMLScoring(
  ScoringData = data,
  ModelObject = NULL,
  ModelType = "mojo",
  H2OShutdown = TRUE,
  MaxMem = "28G",
  NThreads = max(1, parallel::detectCores()-2),
  JavaOptions = '-Xmx1g -XX:ReservedCodeCacheSize=256m',
  ModelPath = normalizePath("./"),
  ModelID = "ModelTest",
  ReturnFeatures = TRUE,
  TransformNumeric = FALSE,
  BackTransNumeric = FALSE,
  TargetColumnName = NULL,
  TransformationObject = NULL,
  TransID = NULL,
  TransPath = NULL,
  MDP_Impute = TRUE,
  MDP_CharToFactor = TRUE,
  MDP_RemoveDates = TRUE,
  MDP_MissFactor = "0",
  MDP_MissNum = -1)

## End(Not run)
```

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 Automated Model Scoring: AutoCatBoostScoring(), AutoH2OMLScoring(), AutoHurdleScoring(), AutoXGBoostScoring(), IntermittentDemandScoringDataGenerator()

Examples

```
## Not run:
# 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)

## End(Not run)
```

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),
  TextType = "individual",
  TextNames = NULL,
  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.

TextType	"Individual" or "Combined" depending on how you build your word2vec models
TextNames	Column names for the text columns to convert to word2vec
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: CatBoostClassifierParams(), CatBoostMultiClassParams(), CatBoostParameterGr
CatBoostRegressionParams(), XGBoostClassifierParams(), XGBoostMultiClassParams(), XGBoostParameterGrids(), XGBoostRegressionMetrics(), XGBoostRegressionParams()

Examples

```
## Not run:
# 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))

## End(Not run)
```

AutoH2OTextPrepScoring

AutoH2OTextPrepScoring is for NLP scoring

Description

This function returns prepared tokenized data for H2O Word2VecModeler scoring

Usage

```
AutoH2OTextPrepScoring(
  data,
  string = NULL,
  MaxMem = NULL,
  NThreads = NULL,
  StartH2O = TRUE
)
```

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
`StartH2O`	Set to TRUE to have H2O start inside this function

Author(s)

Adrian Antico

See Also

Other Misc: `ChartTheme()`, `PrintToPDF()`, `RPM_Binomial_Bandit()`, `tokenizeH2O()`

Examples

```
## Not run:
data <- AutoH2OTextPrepScoring(data = x,
                               string = "text_column",
                               MaxMem = "28G",
                               NThreads = 8,
                               StartH2O = TRUE)

## End(Not run)
```

`AutoHierarchicalFourier`

AutoHierarchicalFourier

Description

AutoHierarchicalFourier reverses the difference

Usage

```
AutoHierarchicalFourier(
  datax = data,
  xRegs = names(XREGS),
  FourierTermS = FourierTerms,
  TimeUniT = TimeUnit,
  FC_PeriodS = FC_Periods,
  TargetColumN = TargetColumn,
  DateColumN = DateColumnName,
  HierarchGroups = NULL,
  IndependentGroups = NULL
)
```

Arguments

datax	data
xRegs	The XREGS
FourierTermS	Number of fourier pairs
TimeUniT	Time unit
FC_PeriodS	Number of forecast periods
TargetColumN	Target column name
DateColumN	Date column name
HierarchGroups	Character vector of categorical columns to fully interact
IndependentGroups	Character vector of categorical columns to run independently

Author(s)

Adrian Antico

See Also

Other Feature Engineering: AutoDataPartition(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), ContinuousTimeDataGen CreateCalendarVariables(), CreateHolidayVariables(), DT_GDL_Feature_Engineering(), DummifyDT(), H2oAutoencoder(), ModelDataPrep(), Partial_DT_GDL_Feature_Engineering(), TimeSeriesFill()

AutoHurdleScoring *AutoHurdleScoring()*

Description

AutoHurdleScoring() can score AutoCatBoostHurdleModel() and AutoXGBoostHurdleModel()

Usage

```
AutoHurdleScoring(
  TestData = NULL,
  Path = NULL,
  ModelID = NULL,
  ModelClass = "catboost",
  ArgList = NULL,
  ModelList = NULL,
  Threshold = NULL
)
```

Arguments

`TestData`	scoring data.table
`Path`	Supply if ArgList is NULL or ModelList is null.
`ModelID`	Supply if ArgList is NULL or ModelList is null. Same as used in model training.
`ModelClass`	Name of model type. "catboost" is currently the only available option
`ArgList`	Output from the hurdle model
`ModelList`	Output from the hurdle model
`Threshold`	NULL to use raw probabilities to predict. Otherwise, supply a threshold

Value

A data.table with the final predicted value, the intermediate model predictions, and your source data

Author(s)

Adrian Antico

See Also

Other Automated Model Scoring: `AutoCatBoostScoring()`, `AutoH2OMLScoring()`, `AutoH2OModeler()`, `AutoXGBoostScoring()`, `IntermittentDemandScoringDataGenerator()`

Examples

```
## Not run:

# XGBoost----

# Define file path
Path <- "C:/Users/aantico/Documents/Package/GUI_Package"

# Create hurdle data with correlated features
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.70,
  N = 25000,
  ID = 3,
  FactorCount = 2L,
  AddDate = TRUE,
  ZIP = 1,
  Classification = FALSE,
  MultiClass = FALSE)
```

```
# Define features
Features <- names(data)[!names(data) %chin%
  c("Adrian","IDcol_1","IDcol_2","IDcol_3","DateTime")]

# Build hurdle model
Output <- RemixAutoML::AutoXGBoostHurdleModel(

  # Operationalization args
  TreeMethod = "hist",
  TrainOnFull = FALSE,
  PassInGrid = NULL,

  # Metadata args
  NThreads = max(1L, parallel::detectCores()-2L),
  ModelID = "ModelTest",
  Paths = normalizePath(Path),
  MetaDataPaths = NULL,
  ReturnModelObjects = TRUE,

  # data args
  data,
  ValidationData = NULL,
  TestData = NULL,
  Buckets = c(0),
  TargetColumnName = "Adrian",
  FeatureColNames = Features,
  IDcols = c("IDcol_1","IDcol_2","IDcol_3"),

  # options
  TransformNumericColumns = NULL,
  SplitRatios = c(0.70, 0.20, 0.10),
  SaveModelObjects = TRUE,
  NumOfParDepPlots = 10L,

  # grid tuning args
  GridTune = FALSE,
  grid_eval_metric = "accuracy",
  MaxModelsInGrid = 1L,
  BaselineComparison = "default",
  MaxRunsWithoutNewWinner = 10L,
  MaxRunMinutes = 60L,

  # bandit hyperparameters
  Trees = 100L,
  eta = seq(0.05,0.40,0.05),
  max_depth = seq(4L, 16L, 2L),

  # random hyperparameters
  min_child_weight = seq(1.0, 10.0, 1.0),
  subsample = seq(0.55, 1.0, 0.05),
  colsample_bytree = seq(0.55, 1.0, 0.05))

# Score XGBoost Hurdle Model
HurdleScores <- RemixAutoML::AutoHurdleScoring(
  TestData = data,
  Path = Path,
```

```
  ModelID = "ModelTest",
  ModelClass = "xgboost",
  ModelList = NULL,
  ArgList = NULL,
  Threshold = NULL)

## End(Not run)
```

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(), H2oIsolationForest(), ResidualOutliers()

Examples

```
## Not run:
data <- data.table::as.data.table(iris)
data <- AutoKMeans(
  data,
  nthreads = 8,
  MaxMem = "28G",
  SaveModels = NULL,
  PathFile = normalizePath("./"),
  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]]
```

```
## End(Not run)
```

AutoLagRollStats *AutoLagRollStats*

Description

AutoLagRollStats Builds lags and a large variety of rolling statistics with options to generate them for hierarchical categorical interactions.

Usage

```
AutoLagRollStats(
  data,
  Targets = NULL,
  HierarchyGroups = NULL,
  IndependentGroups = NULL,
  DateColumn = NULL,
  TimeUnit = "day",
  TimeUnitAgg = "day",
  TimeGroups = "day",
  TimeBetween = NULL,
  RollOnLag1 = TRUE,
  Type = "Lag",
  SimpleImpute = TRUE,
  Lags = c(1:5),
  MA_RollWindows = c(2, 5, 10),
  SD_RollWindows = c(5, 10),
  Skew_RollWindows = c(5, 10),
  Kurt_RollWindows = c(5, 10),
  Quantile_RollWindows = c(10),
  Quantiles_Selected = c("q25", "q75"),
  Debug = FALSE
)
```

Arguments

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

`Targets` A character vector of the column names for the reference column in which you will build your lags and rolling stats

`HierarchyGroups` A vector of categorical column names that you want to have generate all lags and rolling stats done for the individual columns and their full set of interactions.

`IndependentGroups` A vector of categorical column names that you want to have run independently of each other. This will mean that no interaction will be done.

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

`TimeUnit` List the time aggregation level for the time between events features, such as "hour", "day", "weeks", "months", "quarter", or "year"

TimeUnitAgg	List the time aggregation of your data that you want to use as a base time unit for your features. E.g. "raw" or "day"
TimeGroups	A vector of TimeUnits indicators to specify any time-aggregated GDL features you want to have returned. E.g. c("raw" (no aggregation is done),"hour", "day","week","month","quarter","year")
TimeBetween	Specify a desired name for features created for time between events. Set to NULL if you don't want time between events features created.
RollOnLag1	Set to FALSE to build rolling stats off of target columns directly or set to TRUE 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
SimpleImpute	Set to TRUE for factor level imputation of "0" and numeric imputation of -1
Lags	A numeric vector of the specific lags you want to have generated. You must include 1 if WindowingLag = 1.
MA_RollWindows	A numeric vector of the specific rolling statistics window sizes you want to utilize in the calculations.
SD_RollWindows	A numeric vector of Standard Deviation rolling statistics window sizes you want to utilize in the calculations.
Skew_RollWindows	A numeric vector of Skewness rolling statistics window sizes you want to utilize in the calculations.
Kurt_RollWindows	A numeric vector of Kurtosis rolling statistics window sizes you want to utilize in the calculations.
Quantile_RollWindows	A numeric vector of Quantile rolling statistics window sizes you want to utilize in the calculations.
Quantiles_Selected	Select from the following c("q5", "q10", "q15", "q20", "q25", "q30", "q35", "q40", "q45", "q50", "q55", "q60"," q65", "q70", "q75", "q80", "q85", "q90", "q95")
Debug	Set to TRUE to get a print of which steps are running

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(), AutoHierarchicalFourier(), AutoLagRollStatsScoring(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), ContinuousTimeDataGen CreateCalendarVariables(), CreateHolidayVariables(), DT_GDL_Feature_Engineering(), DummifyDT(), H2oAutoencoder(), ModelDataPrep(), Partial_DT_GDL_Feature_Engineering(), TimeSeriesFill()

Examples

```
## Not run:
# Create fake Panel Data----
Count <- 1L
for(Level in LETTERS) {
  datatemp <- RemixAutoML::FakeDataGenerator(
    Correlation = 0.75,
    N = 25000L,
    ID = 0L,
    ZIP = 0L,
    FactorCount = 0L,
    AddDate = TRUE,
    Classification = FALSE,
    MultiClass = FALSE)
  datatemp[, Factor1 := eval(Level)]
  if(Count == 1L) {
    data <- data.table::copy(datatemp)
  } else {
    data <- data.table::rbindlist(
      list(data, data.table::copy(datatemp)))
  }
  Count <- Count + 1L
}

# Add scoring records
data <- RemixAutoML::AutoLagRollStats(

  # Data
  data                 = data,
  DateColumn           = "DateTime",
  Targets              = "Adrian",
  HierarchyGroups      = NULL,
  IndependentGroups    = c("Factor1"),
  TimeUnitAgg          = "days",
  TimeGroups           = c("days", "weeks",
                           "months", "quarters"),
  TimeBetween          = NULL,
  TimeUnit             = "days",

  # Services
  RollOnLag1           = TRUE,
  Type                 = "Lag",
  SimpleImpute         = TRUE,

  # Calculated Columns
  Lags                 = list("days" = c(seq(1,5,1)),
                              "weeks" = c(seq(1,3,1)),
                              "months" = c(seq(1,2,1)),
                              "quarters" = c(seq(1,2,1))),
  MA_RollWindows       = list("days" = c(seq(1,5,1)),
                              "weeks" = c(seq(1,3,1)),
                              "months" = c(seq(1,2,1)),
                              "quarters" = c(seq(1,2,1))),
  SD_RollWindows       = NULL,
  Skew_RollWindows     = NULL,
  Kurt_RollWindows     = NULL,
```

```
    Quantile_RollWindows = NULL,
    Quantiles_Selected   = NULL,
    Debug                = FALSE)

## End(Not run)
```

AutoLagRollStatsScoring *AutoLagRollStatsScoring*

Description

AutoLagRollStatsScoring Builds lags and a large variety of rolling statistics with options to generate them for hierarchical categorical interactions.

Usage

```
AutoLagRollStatsScoring(
  data,
  RowNumsID = "temp",
  RowNumsKeep = 1,
  Targets = NULL,
  HierarchyGroups = NULL,
  IndependentGroups = NULL,
  DateColumn = NULL,
  TimeUnit = "day",
  TimeUnitAgg = "day",
  TimeGroups = "day",
  TimeBetween = NULL,
  RollOnLag1 = 1,
  Type = "Lag",
  SimpleImpute = TRUE,
  Lags = NULL,
  MA_RollWindows = NULL,
  SD_RollWindows = NULL,
  Skew_RollWindows = NULL,
  Kurt_RollWindows = NULL,
  Quantile_RollWindows = NULL,
  Quantiles_Selected = NULL,
  Debug = FALSE
)
```

Arguments

`data`	A data.table you want to run the function on
`RowNumsID`	The name of your column used to id the records so you can specify which rows to keep
`RowNumsKeep`	The RowNumsID numbers that you want to keep
`Targets`	A character vector of the column names for the reference column in which you will build your lags and rolling stats

HierarchyGroups
: A vector of categorical column names that you want to have generate all lags and rolling stats done for the individual columns and their full set of interactions.

IndependentGroups
: Only supply if you do not want HierarchyGroups. A vector of categorical column names that you want to have run independently of each other. This will mean that no interaction will be done.

DateColumn
: The column name of your date column used to sort events over time

TimeUnit
: List the time aggregation level for the time between events features, such as "hour", "day", "weeks", "months", "quarter", or "year"

TimeUnitAgg
: List the time aggregation of your data that you want to use as a base time unit for your features. E.g. "day",

TimeGroups
: A vector of TimeUnits indicators to specify any time-aggregated GDL features you want to have returned. E.g. c("hour", "day","week","month","quarter","year"). STILL NEED TO ADD these '1min', '5min', '10min', '15min', '30min', '45min'

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

RollOnLag1
: Set to FALSE to build rolling stats off of target columns directly or set to TRUE 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

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

Lags
: A numeric vector of the specific lags you want to have generated. You must include 1 if WindowingLag = 1.

MA_RollWindows
: A numeric vector of the specific rolling statistics window sizes you want to utilize in the calculations.

SD_RollWindows
: A numeric vector of Standard Deviation rolling statistics window sizes you want to utilize in the calculations.

Skew_RollWindows
: A numeric vector of Skewness rolling statistics window sizes you want to utilize in the calculations.

Kurt_RollWindows
: A numeric vector of Kurtosis rolling statistics window sizes you want to utilize in the calculations.

Quantile_RollWindows
: A numeric vector of Quantile rolling statistics window sizes you want to utilize in the calculations.

Quantiles_Selected
: Select from the following c("q5", "q10", "q15", "q20", "q25", "q30", "q35", "q40", "q45", "q50", "q55", "q60"," q65", "q70", "q75", "q80", "q85", "q90", "q95")

Debug
: Set to TRUE to get a print out of which step you are on

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(), AutoHierarchicalFourier(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), ContinuousTimeDataGen CreateCalendarVariables(), CreateHolidayVariables(), DT_GDL_Feature_Engineering(), DummifyDT(), H2oAutoencoder(), ModelDataPrep(), Partial_DT_GDL_Feature_Engineering(), TimeSeriesFill()

Examples

```
# Create fake Panel Data----
Count <- 1L
for(Level in LETTERS) {
  datatemp <- RemixAutoML::FakeDataGenerator(
    Correlation = 0.75,
    N = 25000L,
    ID = 0L,
    ZIP = 0L,
    FactorCount = 0L,
    AddDate = TRUE,
    Classification = FALSE,
    MultiClass = FALSE)
  datatemp[, Factor1 := eval(Level)]
  if(Count == 1L) {
    data <- data.table::copy(datatemp)
  } else {
    data <- data.table::rbindlist(
      list(data, data.table::copy(datatemp)))
  }
  Count <- Count + 1L
}

# Create ID columns to know which records to score
data[, ID := .N:1L, by = "Factor1"]
data.table::set(data, i = which(data[["ID"]] == 2L), j = "ID", value = 1L)

# Score records
data <- RemixAutoML::AutoLagRollStatsScoring(

  # Data
  data                 = data,
  RowNumsID            = "ID",
  RowNumsKeep          = 1,
  DateColumn           = "DateTime",
  Targets              = "Adrian",
  HierarchyGroups      = c("Store","Dept"),
  IndependentGroups    = NULL,

  # Services
  TimeBetween          = NULL,
  TimeGroups           = c("days", "weeks", "months"),
  TimeUnit             = "day",
  TimeUnitAgg          = "day",
```

```
RollOnLag1           = TRUE,
Type                 = "Lag",
SimpleImpute         = TRUE,

# Calculated Columns
Lags                  = list("days" = c(seq(1,5,1)),
                             "weeks" = c(seq(1,3,1)),
                             "months" = c(seq(1,2,1))),
MA_RollWindows        = list("days" = c(seq(1,5,1)),
                             "weeks" = c(seq(1,3,1)),
                             "months" = c(seq(1,2,1))),
SD_RollWindows        = list("days" = c(seq(1,5,1)),
                             "weeks" = c(seq(1,3,1)),
                             "months" = c(seq(1,2,1))),
Skew_RollWindows      = list("days" = c(seq(1,5,1)),
                             "weeks" = c(seq(1,3,1)),
                             "months" = c(seq(1,2,1))),
Kurt_RollWindows      = list("days" = c(seq(1,5,1)),
                             "weeks" = c(seq(1,3,1)),
                             "months" = c(seq(1,2,1))),
Quantile_RollWindows  = list("days" = c(seq(1,5,1)),
                             "weeks" = c(seq(1,3,1)),
                             "months" = c(seq(1,2,1))),
Quantiles_Selected    = c("q5","q10","q95"),
Debug                 = FALSE)
```

`AutoLimeAid` *AutoLimeAid automated lime*

Description

AutoLimeAid automated lime explanations and lime model builds.

Usage

```
AutoLimeAid(
  EvalPredsData = data,
  LimeTrainingData = data,
  LimeBins = 10,
  LimeIterations = 7500,
  LimeNumFeatures = 0,
  LimeModel = NULL,
  LimeModelPath = NULL,
  LimeModelID = NULL,
  MLModel = NULL,
  MLModelPath = NULL,
  MLMetaDataPath = NULL,
  MLModelID = NULL,
  ModelType = "xgboost",
  TargetType = "classification",
  NThreads = parallel::detectCores(),
  MaxMem = "32G",
```

```
  FeatureColumnNames = TestModel$ColNames,
  IDcols = NULL,
  FactorLevelsList = TestModel$FactorLevels,
  TargetLevels = NULL,
  OneHot = FALSE,
  ReturnFeatures = TRUE,
  TransformNumeric = FALSE,
  BackTransNumeric = FALSE,
  TargetColumnName = NULL,
  TransformationObject = NULL,
  TransID = NULL,
  TransPath = NULL,
  MDP_Impute = TRUE,
  MDP_CharToFactor = TRUE,
  MDP_RemoveDates = TRUE,
  MDP_MissFactor = "0",
  MDP_MissNum = -1
)
```

Arguments

EvalPredsData	Data used for interpretation. Should be the same kind of data used on ML_Scoring functions.
LimeTrainingData	Data used to train your ML model
LimeBins	Number of bins to use for bucketing numeric variables
LimeIterations	Number of lime permutations ran to generate interpretation of predicted value
LimeNumFeatures	How many features do you want to be considering for the Lime evaluation? Set to 0 to use all features
LimeModel	Supply a model if you have one available. Otherwise, provide a model path and either it will be pulling in or made and saved there.
LimeModelPath	Supply a path to where your model is located or to be stored.
LimeModelID	Provide a name for your model. If left NULL, a name will be created for you (and a new model).
MLModel	Supply the model object (except for H2O models). Can leave null.
MLModelPath	Supply a path to where your model is located. If this is supplied, the model will be pulled in from file (even if you supply a model)
MLMetaDataPath	Supply a path to where your model metadata is located (might be the same of the MLModelPath). If this is supplied, artifacts about the model will be pulled in from there.
MLModelID	The name of your model as read in the file directory
ModelType	Choose from "xgboost", "h2o", "catboost"
TargetType	For catboost models only. Select from "classification", "regression", "multiclass"
NThreads	Number of CPU threads.
MaxMem	Set the max memory you want to allocate. E.g. "32G"

FeatureColumnNames
: The names of the features used in training your ML model (should be returned with the model or saved to file)

IDcols
: The ID columns used in either CatBoost or XGBoost

FactorLevelsList
: = TestModel$FactorLevels,

TargetLevels
: The target levels used in MultiClass models

OneHot
: Replicate what you did with the model training

ReturnFeatures
: TRUE or FALSE

TransformNumeric
: Replicate what you did with the model training

BackTransNumeric
: TRUE or FALSE. Replicate what you did with the model training.

TargetColumnName
: For the transformations

TransformationObject
: TRUE or FALSE. Replicate what you did with the model training.

TransID
: Set to the ID used in model training.

TransPath
: Same path used in model training.

MDP_Impute
: Replicate what you did with the model training.

MDP_CharToFactor
: Replicate what you did with the model training.

MDP_RemoveDates
: Replicate what you did with the model training.

MDP_MissFactor
: Replicate what you did with the model training.

MDP_MissNum
: Replicate what you did with the model training.

Value

LimeModelObject and Lime Explanations

Author(s)

Adrian Antico

See Also

Other Model Evaluation and Interpretation: EvalPlot(), LimeModel(), ParDepCalPlots(), RedYellowGreen(), threshOptim()

Examples

```
## Not run:
# CatBoost data generator
dataGenH2O <- function() {
  Correl <- 0.85
  N <- 10000
  data <- data.table::data.table(Classification = runif(N))
  data[, x1 := qnorm(Classification)]
  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[, Classification := ifelse(Classification > 0.5, 1, 0)]
  rm(N,Correl)
  return(data)
}
data <- dataGenH2O()
TestModel <- RemixAutoML::AutoCatBoostRegression(
  data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = "Classification",
  FeatureColNames = c(2:12),
  PrimaryDateColumn = NULL,
  IDcols = NULL,
  MaxModelsInGrid = 3,
  task_type = "GPU",
  eval_metric = "RMSE",
  Trees = 50,
  GridTune = FALSE,
  model_path = "C:/Users/aantico/Documents/Package/GUI_Package",
  metadata_path = NULL,
  ModelID = "Adrian",
  NumOfParDepPlots = 15,
  ReturnModelObjects = TRUE,
  SaveModelObjects = TRUE,
  PassInGrid = NULL)

# CatBoost Build Lime Model and Explanations
LimeOutput <- RemixAutoML::AutoLimeAid(
  EvalPredsData = data[c(1,15)],
  LimeTrainingData = data,
  LimeBins = 10,
  LimeIterations = 7500,
  LimeNumFeatures = 0,
  TargetType = "regression",
  LimeModel = NULL,
  LimeModelPath = "C:/Users/aantico/Documents/Package/GUI_Package",
  LimeModelID = "AdrianLime",
  MLModel = NULL,
  MLModelPath = "C:/Users/aantico/Documents/Package/GUI_Package",
  MLMetaDataPath = NULL,
  MLModelID = "Adrian",
```

```
  ModelType = "catboost",
  NThreads = parallel::detectCores(),
  MaxMem = "14G",
  FeatureColumnNames = NULL,
  IDcols = NULL,
  FactorLevelsList = NULL,
  TargetLevels = NULL,
  OneHot = FALSE,
  ReturnFeatures = TRUE,
  TransformNumeric = FALSE,
  BackTransNumeric = FALSE,
  TargetColumnName = NULL,
  TransformationObject = NULL,
  TransID = NULL,
  TransPath = NULL,
  MDP_Impute = TRUE,
  MDP_CharToFactor = TRUE,
  MDP_RemoveDates = TRUE,
  MDP_MissFactor = "0",
  MDP_MissNum = -1)

# Plot lime objects
lime::plot_features(LimeOutput$LimeExplanations)
suppressWarnings(lime::plot_explanations(LimeOutput$LimeExplanations))

# H2O data generator
dataGenH2O <- function() {
  Correl <- 0.85
  N <- 10000
  data <- data.table::data.table(Classification = runif(N))
  data[, x1 := qnorm(Classification)]
  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[, Classification := ifelse(Classification > 0.5, 1, 0)]
  rm(N,Correl)
  return(data)
}
data <- dataGenH2O()
TestModel <- RemixAutoML::AutoH2oDRFClassifier(
  data = data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
```

```
  TargetColumnName = "Classification",
  FeatureColNames = setdiff(names(data),"Classification"),
  eval_metric = "auc",
  Trees = 50,
  GridTune = FALSE,
  MaxMem = "32G",
  NThreads = max(1, parallel::detectCores()-2),
  MaxModelsInGrid = 10,
  model_path = "C:/Users/aantico/Desktop/Retention Analytics",
  metadata_path = NULL,
  ModelID = "Adrian",
  NumOfParDepPlots = 10,
  ReturnModelObjects = TRUE,
  SaveModelObjects = TRUE,
  IfSaveModel = "standard",
  H2OShutdown = TRUE)

LimeOutput <- RemixAutoML::AutoLimeAid(
  EvalPredsData = data[c(1,15)],
  LimeTrainingData = data,
  LimeBins = 10,
  LimeIterations = 7500,
  TargetType = "regression",
  LimeNumFeatures = 0,
  LimeModel = NULL,
  LimeModelPath = "C:/Users/aantico/Desktop/Retention Analytics",
  LimeModelID = "AdrianLime",
  MLModel = NULL,
  MLModelPath = "C:/Users/aantico/Desktop/Retention Analytics",
  MLMetaDataPath = NULL,
  MLModelID = "Adrian",
  ModelType = "h2o",
  NThreads = parallel::detectCores(),
  MaxMem = "14G",
  FeatureColumnNames = NULL,
  IDcols = NULL,
  FactorLevelsList = NULL,
  TargetLevels = NULL,
  OneHot = FALSE,
  ReturnFeatures = TRUE,
  TransformNumeric = FALSE,
  BackTransNumeric = FALSE,
  TargetColumnName = NULL,
  TransformationObject = NULL,
  TransID = NULL,
  TransPath = NULL,
  MDP_Impute = TRUE,
  MDP_CharToFactor = TRUE,
  MDP_RemoveDates = TRUE,
  MDP_MissFactor = "0",
  MDP_MissNum = -1)

# Plot lime objects
lime::plot_features(LimeOutput$LimeExplanations)
suppressWarnings(lime::plot_explanations(LimeOutput$LimeExplanations))

# XGBoost create data function
```

```
dataGenXGBoost <- function() {
  Correl <- 0.85
  N <- 10000
  data <- data.table::data.table(Classification = runif(N))
  data[, x1 := qnorm(Classification)]
  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[, Classification := ifelse(Classification > 0.5, 1, 0)]
  rm(Correl,N)
  return(data)
}
data <- dataGenXGBoost()
TestModel <- RemixAutoML::AutoXGBoostClassifier(
  data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = "Classification",
  FeatureColNames = 2:12,
  IDcols = NULL,
  eval_metric = "auc",
  Trees = 50,
  GridTune = FALSE,
  grid_eval_metric = "auc",
  MaxModelsInGrid = 10,
  NThreads = 8,
  TreeMethod = "hist",
  model_path = "C:/Users/aantico/Desktop/Retention Analytics",
  metadata_path = NULL,
  ModelID = "Adrian2",
  NumOfParDepPlots = 3,
  ReturnModelObjects = TRUE,
  ReturnFactorLevels = TRUE,
  SaveModelObjects = TRUE,
  PassInGrid = NULL)

# XGBoost Build Lime and Generate Output
LimeOutput <- RemixAutoML::AutoLimeAid(
  EvalPredsData = data[c(1,15)],
  LimeTrainingData = data,
  LimeBins = 10,
  TargetType = "classification",
  LimeIterations = 7500,
```

```
  LimeNumFeatures = 0,
  LimeModel = NULL,
  LimeModelPath = "C:/Users/aantico/Desktop/Retention Analytics",
  LimeModelID = "Adrian2Lime",
  MLModel = NULL,
  MLModelPath = "C:/Users/aantico/Desktop/Retention Analytics",
  MLMetaDataPath = NULL,
  MLModelID = "Adrian2",
  ModelType = "xgboost",
  NThreads = parallel::detectCores(),
  MaxMem = "14G",
  FeatureColumnNames = NULL,
  IDcols = NULL,
  FactorLevelsList = NULL,
  TargetLevels = NULL,
  OneHot = FALSE,
  ReturnFeatures = TRUE,
  TransformNumeric = FALSE,
  BackTransNumeric = FALSE,
  TargetColumnName = NULL,
  TransformationObject = NULL,
  TransID = NULL,
  TransPath = NULL,
  MDP_Impute = TRUE,
  MDP_CharToFactor = TRUE,
  MDP_RemoveDates = TRUE,
  MDP_MissFactor = "0",
  MDP_MissNum = -1)

# Plot lime objects
lime::plot_features(LimeOutput$LimeExplanations)
suppressWarnings(lime::plot_explanations(LimeOutput$LimeExplanations))

## End(Not run)
```

AutoMarketBasketModel *AutoMarketBasketModel function runs a market basket analysis automatically*

Description

AutoMarketBasketModel function runs a market basket analysis automatically. It will convert your data, run the algorithm, and add on additional significance values not orginally contained within.

Usage

```
AutoMarketBasketModel(
  data,
  OrderIDColumnName,
  ItemIDColumnName,
  LHS_Delimeter = ",",
  Support = 0.001,
  Confidence = 0.1,
  MaxLength = 2,
```

```
  MinLength = 2,
  MaxTime = 5
)
```

Arguments

data
: This is your transactions data set

OrderIDColumnName
: Supply your column name for the Order ID Values

ItemIDColumnName
: Supply your column name for the Item ID Values

LHS_Delimeter
: Default delimeter for separating multiple ItemID's is a comma.

Support
: Threshold for inclusion using support

Confidence
: Threshold for inclusion using confidence

MaxLength
: Maximum combinations of Item ID (number of items in basket to consider)

MinLength
: Minimum length of combinations of ItemID (number of items in basket to consider)

MaxTime
: Max run time per iteration (default is 5 seconds)

Author(s)

Adrian Antico and Douglas Pestana

See Also

Chi-sq statistics and p-values based on this paper: http://www.cs.bc.edu/~alvarez/ChiSquare/chi2tr.pdf

Examples

```
## Not run:
rules_data <- AutoMarketBasketModel(
  data,
  OrderIDColumnName = "OrderNumber",
  ItemIDColumnName = "ItemNumber",
  LHS_Delimeter = ",",
  Support = 0.001,
  Confidence = 0.1,
  MaxLength = 2,
  MinLength = 2,
  MaxTime = 5)

## End(Not run)
```

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

Examples

```
## Not run:
# 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

## End(Not run)
```

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 Recommenders: AutoRecommenderScoring(), AutoRecommender()

Examples

```
## Not run:
RatingsMatrix <- AutoRecomDataCreate(
  data,
  EntityColName = "CustomerID",
  ProductColName = "StockCode",
  MetricColName = "TotalSales",
  ReturnMatrix = TRUE)

## End(Not run)
```

AutoRecommender *Automatically build the best recommender model among models available.*

Description

This function returns the winning model that you pass onto AutoRecommenderScoring

Usage

```
AutoRecommender(
  data,
  Partition = "Split",
  KFolds = 1,
  Ratio = 0.75,
  Given = 1,
```

```
  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 1 for traditional train and test. Choose greater than 1 for the number of cross validations
`Ratio`	The ratio for train and test. E.g. 0.75 for 75 percent data allocated to training
`Given`	The number of products you would like to evaluate. Negative values implement all-but schemes.
`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 Recommenders: `AutoRecomDataCreate()`, `AutoRecommenderScoring()`

Examples

```
## Not run:
WinningModel <- AutoRecommender(
  RatingsMatrix,
  Partition = "Split",
  KFolds = 1,
  Ratio = 0.75,
  Given = 1,
  RatingType = "TopN",
  RatingsKeep = 20,
  SkipModels = "AssociationRules",
  ModelMetric = "TPR")

## End(Not run)
```

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",
  NumItemsReturn = 1
)
```

Arguments

data	The binary ratings matrix from RecomDataCreate()
WinningModel	The winning model returned from AutoRecommender()
EntityColName	Typically your customer ID
ProductColName	Something like "StockCode"
NumItemsReturn	Number of items to return on scoring

Value

Returns the prediction data

Author(s)

Adrian Antico and Douglas Pestana

See Also

Other Recommenders: AutoRecomDataCreate(), AutoRecommender()

Examples

```
## Not run:
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")

## End(Not run)
```

AutoTBATS *AutoTBATS*

Description

AutoTBATS is a multi-armed bandit model testing framework for AR and SAR NNets. Randomized probability matching is the underlying bandit algorithm. Model evaluation is done by blending the training error and the validation error from testing the model on out of sample data. The bandit algorithm compares the performance of the current build against the previous builds which starts with the classic nnetar model from the forecast package. Depending on how many lags, seasonal lags, and fourier pairs you test the number of combinations of features to test begins to approach 10,000 different combinations of settings. The function tests out transformations, differencing, and variations of the lags, seasonal lags, and fourier pairs. The paramter space is broken up into various buckets that are increasing in sophistication. The bandit algorithm samples from those buckets and based on many rounds of testing it determines which buckets to generate samples from more frequently based on the models performance coming from that bucket. All of the models have performance data collected on them and a final rebuild is initiated when a winner is found. The rebuild process begins by retraining the model with the settings that produced the best performance. If the model fails to build, for whatever reason, the next best buildable model is rebuilt.

Usage

```
AutoTBATS(
  data,
  TargetVariableName,
  DateColumnName,
  TimeAggLevel = "week",
  EvaluationMetric = "MAE",
  NumHoldOutPeriods = 5L,
  NumFCPeriods = 5L,
  MaxLags = 5L,
  MaxMovingAverages = 5L,
  MaxSeasonalPeriods = 1L,
  TrainWeighting = 0.5,
  MaxConsecutiveFails = 12L,
  MaxNumberModels = 100L,
  MaxRunTimeMinutes = 10L
)
```

Arguments

- `data` Source data.table
- `TargetVariableName` Name of your time series target variable
- `DateColumnName` Name of your date column
- `TimeAggLevel` Choose from "year", "quarter", "month", "week", "day", "hour"
- `EvaluationMetric` Choose from MAE, MSE, and MAPE
- `NumHoldOutPeriods` Number of time periods to use in the out of sample testing
- `NumFCPeriods` Number of periods to forecast
- `MaxLags` A single value of the max number of lags to use in the internal auto.arima of tbats
- `MaxMovingAverages` A single value of the max number of moving averages to use in the internal auto.arima of tbats
- `MaxSeasonalPeriods` A single value for the max allowable seasonal periods to be tested in the tbats framework
- `TrainWeighting` Model ranking is based on a weighted average of training metrics and out of sample metrics. Supply the weight of the training metrics, such as 0.50 for 50 percent.
- `MaxConsecutiveFails` When a new best model is found MaxConsecutiveFails resets to zero. Indicated the number of model attemps without a new winner before terminating the procedure.
- `MaxNumberModels` Indicate the maximum number of models to test.
- `MaxRunTimeMinutes` Indicate the maximum number of minutes to wait for a result.

Author(s)

Adrian Antico

See Also

Other Automated Time Series: `AutoBanditNNet()`, `AutoBanditSarima()`, `AutoCatBoostFreqSizeScoring()`, `AutoH2oGBMFreqSizeScoring()`, `AutoTS()`

`AutoTransformationCreate`

AutoTransformationCreate is a function for automatically identifying the optimal transformations for numeric features and transforming them once identified.

Description

AutoTransformationCreate is a function for automatically identifying the optimal transformations for numeric features and transforming them once identified. This function will loop through your selected transformation options (YeoJohnson, BoxCox, Asinh, Asin, and Logit) and find the one that produces data that is the closest to normally distributed data. It then makes the transformation and collects the metadata information for use in the AutoTransformationScore() function, either by returning the objects (always) or saving them to file (optional).

Usage

```
AutoTransformationCreate(
  data,
  ColumnNames = NULL,
 Methods = c("BoxCox", "YeoJohnson", "Asinh", "Log", "LogPlus1", "Asin", "Logit",
    "Identity"),
  Path = NULL,
  TransID = "ModelID",
  SaveOutput = FALSE
)
```

Arguments

`data`	This is your source data
`ColumnNames`	List your columns names in a vector, for example, c("Target", "IV1")
`Methods`	Choose from "YeoJohnson", "BoxCox", "Asinh", "Log", "LogPlus1", "Asin", "Logit", and "Identity".
`Path`	Set to the directly where you want to save all of your modeling files
`TransID`	Set to a character value that corresponds with your modeling project
`SaveOutput`	Set to TRUE to save necessary file to run AutoTransformationScore()

Value

data with transformed columns and the transformation object for back-transforming later

Author(s)

Adrian Antico

See Also

Other Feature Engineering: `AutoDataPartition()`, `AutoHierarchicalFourier()`, `AutoLagRollStatsScoring()`, `AutoLagRollStats()`, `AutoTransformationScore()`, `AutoWord2VecModeler()`, `ContinuousTimeDataGenerator()`, `CreateCalendarVariables()`, `CreateHolidayVariables()`, `DT_GDL_Feature_Engineering()`, `DummifyDT()`, `H2oAutoencoder()`, `ModelDataPrep()`, `Partial_DT_GDL_Feature_Engineering()`, `TimeSeriesFill()`

Examples

```
## Not run:
Correl <- 0.85
N <- 1000
data <- data.table::data.table(Adrian = runif(N))
```

```
data[, x1 := qnorm(Adrian)]
data[, x2 := runif(N)]
data[, Adrian1 := log(pnorm(Correl * x1 + sqrt(1-Correl^2) * qnorm(x2)))]
data <- RemixAutoML::AutoTransformationCreate(
   data,
   ColumnNames = "Sample",
   Methods = c("BoxCox",
               "YeoJohnson",
               "Asinh",
               "Log",
               "LogPlus1",
               "Asin",
               "Logit",
               "Identity"),
   Path = NULL,
   TransID = "Trans",
   SaveOutput = FALSE)

## End(Not run)
```

`AutoTransformationScore` *AutoTransformationScore() is a the complimentary function to AutoTransformationCreate()*

Description

AutoTransformationScore() is a the compliment function to AutoTransformationCreate(). Automatically apply or inverse the transformations you identified in AutoTransformationCreate() to other data sets. This is useful for applying transformations to your validation and test data sets for modeling. It's also useful for back-transforming your target and prediction columns after you have build and score your models so you can obtain statistics on the original features.

Usage

```
AutoTransformationScore(
  ScoringData,
  FinalResults,
  Type = "Inverse",
  TransID = "TestModel",
  Path = NULL
)
```

Arguments

`ScoringData`	This is your source data
`FinalResults`	This is the FinalResults output object from AutoTransformationCreate().
`Type`	Set to "Inverse" to back-transfrom or "Apply" for applying the transformation.
`TransID`	Set to a character value that corresponds with your modeling project
`Path`	Set to the directly where you want to save all of your modeling files

Value

data with transformed columns

Author(s)

Adrian Antico

See Also

Other Feature Engineering: AutoDataPartition(), AutoHierarchicalFourier(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoWord2VecModeler(), ContinuousTimeDataGenerator(), CreateCalendarVariables(), CreateHolidayVariables(), DT_GDL_Feature_Engineering(), DummifyDT(), H2oAutoencoder(), ModelDataPrep(), Partial_DT_GDL_Feature_Engineering(), TimeSeriesFill()

Examples

```
## Not run:
Correl <- 0.85
N <- 1000
data <- data.table::data.table(Adrian = runif(N))
data[, x1 := qnorm(Adrian)]
data[, x2 := runif(N)]
data[, Adrian1 := log(pnorm(Correl * x1 + sqrt(1-Correl^2) * qnorm(x2)))]
data <- RemixAutoML::AutoTransformationScore(
   data,
   FinalResults,
   Path = NULL,
   TransID = "Trans")

## End(Not run)
```

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",
  InnerEval = "AICc",
  TimeUnit = "day",
  Lags = 25,
  SLags = 2,
  MaxFourierPairs = 0,
  NumCores = 4,
  SkipModels = NULL,
  StepWise = TRUE,
  TSClean = TRUE,
  ModelFreq = TRUE,
  PrintUpdates = FALSE,
  PlotPredictionIntervals = TRUE
)
```

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"
`InnerEval`	Choose from AICC, AIC, and BIC. These are what the time series models use internally to optimize
`TimeUnit`	is the level of aggregation your dataset comes in. Choices include: hour, day, week, month, quarter, year, 1Min, 5Min, 10Min, 15Min, and 30Min
`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)
`MaxFourierPairs`	Set the max number of Fourier terms to test out. They will be utilized in the ARIMA and NN models.
`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
`PlotPredictionIntervals`	Set to FALSE to not print prediction intervals on your plot output

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 with 80

Author(s)

Adrian Antico and Douglas Pestana

See Also

Other Automated Time Series: AutoBanditNNet(), AutoBanditSarima(), AutoCatBoostFreqSizeScoring(), AutoH2oGBMFreqSizeScoring(), AutoTBATS()

Examples

```
## Not run:
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",
  InnerEval               = "AICc",
  TimeUnit                = "day",
  Lags                    = 1,
  SLags                   = 1,
  MaxFourierPairs         = 0,
  NumCores                = 4,
```

```
  SkipModels              = c("NNET","TBATS","ETS",
    "TSLM","ARFIMA","DSHW"),
  StepWise                = TRUE,
  TSClean                 = FALSE,
  ModelFreq               = TRUE,
  PlotPredictionIntervals = TRUE,
  PrintUpdates            = FALSE)
ForecastData <- output$Forecast
ModelEval    <- output$EvaluationMetrics
WinningModel <- output$TimeSeriesModel

## End(Not run)
```

`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,
  BuildType = "Combined",
  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
`BuildType`	Choose from "individual" or "combined". Individual will build a model for every text column. Combined will build a single model for all columns.
`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()`, `AutoHierarchicalFourier()`, `AutoLagRollStatsScoring()`, `AutoLagRollStats()`, `AutoTransformationCreate()`, `AutoTransformationScore()`, `ContinuousTimeDataGenera` `CreateCalendarVariables()`, `CreateHolidayVariables()`, `DT_GDL_Feature_Engineering()`, `DummifyDT()`, `H2oAutoencoder()`, `ModelDataPrep()`, `Partial_DT_GDL_Feature_Engineering()`, `TimeSeriesFill()`

Examples

```
## Not run:
data <- AutoWord2VecModeler(
  data,
  BuildType = "individual",
  stringCol = c("Text_Col1","Text_Col2"),
  KeepStringCol = FALSE,
  model_path = normalizePath("./"),
  vects = 100,
  SaveStopWords = FALSE,
  MinWords = 1,
  WindowSize = 1,
  Epochs = 25,
  StopWords = NULL,
  SaveModel = "standard",
  Threads = max(1,parallel::detectCores()-2),
  MaxMemory = "28G",
  SaveOutput = TRUE)

## End(Not run)
```

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()

Examples

```
## Not run:
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"))

## End(Not run)
```

AutoXGBoostCARMA — *AutoXGBoostCARMA Automated XGBoost Calendar, Holiday, ARMA, and Trend Variables Forecasting*

Description

AutoXGBoostCARMA Automated XGBoost Calendar, Holiday, ARMA, and Trend Variables Forecasting. Create hundreds of thousands of time series forecasts using this function.

Usage

```
AutoXGBoostCARMA(
  data,
  NonNegativePred = FALSE,
  RoundPreds = FALSE,
  TrainOnFull = FALSE,
  TargetColumnName = NULL,
  DateColumnName = NULL,
  HierarchGroups = NULL,
  GroupVariables = NULL,
  FC_Periods = 5,
  TimeUnit = "week",
  TimeGroups = c("weeks", "months"),
  TargetTransformation = FALSE,
 Methods = c("BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson"),
  AnomalyDetection = NULL,
  XREGS = NULL,
  Lags = c(1:5),
  MA_Periods = c(1:5),
  SD_Periods = NULL,
  Skew_Periods = NULL,
```

```
  Kurt_Periods = NULL,
  Quantile_Periods = NULL,
  Quantiles_Selected = NULL,
  Difference = TRUE,
  FourierTerms = 6,
  CalendarVariables = c("second", "minute", "hour", "wday", "mday", "yday", "week",
    "isoweek", "month", "quarter", "year"),
  HolidayVariable = c("USPublicHolidays", "EasterGroup", "ChristmasGroup",
    "OtherEcclesticalFeasts"),
  HolidayLags = 1L,
  HolidayMovingAverages = 3L,
  TimeTrendVariable = FALSE,
  DataTruncate = FALSE,
  ZeroPadSeries = NULL,
  SplitRatios = c(1 - 10/100, 10/100),
  TreeMethod = "hist",
  NThreads = max(1, parallel::detectCores() - 2L),
  EvalMetric = "MAE",
  GridTune = FALSE,
  GridEvalMetric = "mae",
  ModelCount = 1L,
  NTrees = 1000L,
  PartitionType = "timeseries",
  Timer = TRUE,
  DebugMode = FALSE
)
```

Arguments

- `data` Supply your full series data set here
- `NonNegativePred` TRUE or FALSE
- `RoundPreds` Rounding predictions to an integer value. TRUE or FALSE. Defaults to FALSE
- `TrainOnFull` Set to TRUE to train on full data
- `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"
- `HierarchGroups` = NULL Character vector or NULL with names of the columns that form the interaction hierarchy
- `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. "1min", "5min", "10min", "15min", "30min", "hour", "day", "week", "month", "quarter", "year"
- `TimeGroups` Select time aggregations for adding various time aggregated GDL features.
- `TargetTransformation` Run AutoTransformationCreate() to find best transformation for the target variable. Tests YeoJohnson, BoxCox, and Asigh (also Asin and Logit for proportion target variables).

`Methods` Transformation options to test which include "BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson"

`AnomalyDetection`
NULL for not using the service. Other, provide a list, e.g. AnomalyDetection = list("tstat_high" = 4, tstat_low = -4)

`XREGS` Additional data to use for model development and forecasting. Data needs to be a complete series which means both the historical and forward looking values over the specified forecast window needs to be supplied.

`Lags` Select the periods for all lag variables you want to create. E.g. c(1:5,52)

`MA_Periods` Select the periods for all moving average variables you want to create. E.g. c(1:5,52)

`SD_Periods` Select the periods for all moving standard deviation variables you want to create. E.g. c(1:5,52)

`Skew_Periods` Select the periods for all moving skewness variables you want to create. E.g. c(1:5,52)

`Kurt_Periods` Select the periods for all moving kurtosis variables you want to create. E.g. c(1:5,52)

`Quantile_Periods`
Select the periods for all moving quantiles variables you want to create. E.g. c(1:5,52)

`Quantiles_Selected`
Select from the following c("q5","q10","q15","q20","q25","q30","q35","q40","q45","q50","q55","q6

`Difference` Set to TRUE to put the I in ARIMA

`FourierTerms` Set to the max number of pairs

`CalendarVariables`
NULL, or select from "second", "minute", "hour", "wday", "mday", "yday", "week", "isoweek", "month", "quarter", "year"

`HolidayVariable`
NULL, or select from "USPublicHolidays", "EasterGroup", "ChristmasGroup", "OtherEcclesticalFeasts"

`HolidayLags` Number of lags for the holiday counts

`HolidayMovingAverages`
Number of moving averages for holiday counts

`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

`ZeroPadSeries` Set to "all", "inner", or NULL. See TimeSeriesFill for explanation

`SplitRatios` E.g c(0.7,0.2,0.1) for train, validation, and test sets

`TreeMethod` Choose from "hist", "gpu_hist"

`NThreads` Set the maximum number of threads you'd like to dedicate to the model run. E.g. 8

`EvalMetric` Select from "r2", "RMSE", "MSE", "MAE"

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
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	Setting to TRUE prints out the forecast number while it is building
DebugMode	Setting to TRUE generates printout of all header code comments during run time of function

Value

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

Author(s)

Adrian Antico

See Also

Other Automated Panel Data Forecasting: AutoCatBoostCARMA(), AutoCatBoostHurdleCARMA(), AutoCatBoostVectorCARMA(), AutoH2OCARMA()

Examples

```
## Not run:

# Load Walmart Data from Dropbox----
data <- data.table::fread(
  "https://www.dropbox.com/s/2str3ek4f4cheqi/walmart_train.csv?dl=1")

# Subset for Stores / Departments With Full Series
data <- data[, Counts := .N, by = c("Store","Dept")][Counts == 143][
  , Counts := NULL]

# Subset Columns (remove IsHoliday column)----
keep <- c("Store","Dept","Date","Weekly_Sales")
data <- data[, ..keep]
data <- data[Store %in% c(1,2)]

xregs <- data.table::copy(data)
xregs[, GroupVar := do.call(paste, c(.SD, sep = " ")), .SDcols = c("Store","Dept")]
xregs[, c("Store","Dept") := NULL]
data.table::setnames(xregs, "Weekly_Sales", "Other")
xregs[, Other := jitter(Other, factor = 25)]
data <- data[as.Date(Date) < as.Date('2012-09-28')]

 # Build forecast
XGBoostResults <- AutoXGBoostCARMA(
```

```
  # Data Artifacts
  data = data,
  NonNegativePred = FALSE,
  RoundPreds = FALSE,
  TargetColumnName = "Weekly_Sales",
  DateColumnName = "Date",
  HierarchGroups = NULL,
  GroupVariables = c("Store","Dept"),
  TimeUnit = "weeks",
  TimeGroups = c("weeks","months"),

  # Data Wrangling Features
  ZeroPadSeries = NULL,
  DataTruncate = FALSE,
  SplitRatios = c(1 - 10 / 138, 10 / 138),
  PartitionType = "timeseries",
  AnomalyDetection = NULL,

  # Productionize
  FC_Periods = 4,
  TrainOnFull = FALSE,
  TreeMethod = "hist",
  EvalMetric = "RMSE",
  GridTune = FALSE,
  ModelCount = 5,
  NThreads = 8,
  Timer = TRUE,
  DebugMode = FALSE,

  # Target Transformations
  TargetTransformation = TRUE,
  Methods = c("BoxCox", "Asinh", "Asin", "Log",
              "LogPlus1","Logit","YeoJohnson"),
  Difference = FALSE,

  # Features
  Lags = list("weeks" = seq(1L, 10L, 1L),
              "months" = seq(1L, 5L, 1L)),
  MA_Periods = list("weeks" = seq(5L, 20L, 5L),
                    "months" = seq(2L, 10L, 2L)),
  SD_Periods = NULL,
  Skew_Periods = NULL,
  Kurt_Periods = NULL,
  Quantile_Periods = NULL,
  HolidayLags = 1,
  HolidayMovingAverages = 1:2,
  Quantiles_Selected = c("q5","q95"),
  XREGS = xregs,
  FourierTerms = 4,
  CalendarVariables = c("week", "month", "quarter"),
  HolidayVariable = c("USPublicHolidays","EasterGroup",
    "ChristmasGroup","OtherEcclesticalFeasts"),
  TimeTrendVariable = TRUE,
  NTrees = 300)

UpdateMetrics <- print(
  XGBoostResults$ModelInformation$EvaluationMetrics[
```

```
    Metric == "MSE", MetricValue := sqrt(MetricValue)])
print(UpdateMetrics)
XGBoostResults$ModelInformation$EvaluationMetricsByGroup[order(-R2_Metric)]
XGBoostResults$ModelInformation$EvaluationMetricsByGroup[order(MAE_Metric)]
XGBoostResults$ModelInformation$EvaluationMetricsByGroup[order(MSE_Metric)]
XGBoostResults$ModelInformation$EvaluationMetricsByGroup[order(MAPE_Metric)]

## End(Not run)
```

`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,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  IDcols = NULL,
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  ReturnFactorLevels = TRUE,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  Verbose = 0L,
  NumOfParDepPlots = 3L,
  NThreads = parallel::detectCores(),
  eval_metric = "auc",
  TreeMethod = "hist",
  GridTune = FALSE,
  BaselineComparison = "default",
  MaxModelsInGrid = 10L,
  MaxRunsWithoutNewWinner = 20L,
  MaxRunMinutes = 24L * 60L,
  PassInGrid = NULL,
  Shuffles = 1L,
  Trees = 1000L,
```

```
    eta = seq(0.05, 0.4, 0.05),
    max_depth = seq(4L, 16L, 2L),
    min_child_weight = seq(1, 10, 1),
    subsample = seq(0.55, 1, 0.05),
    colsample_bytree = seq(0.55, 1, 0.05)
  )
```

Arguments

data	This is your data set for training and testing your model
TrainOnFull	Set to TRUE to train on full data
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.
model_path	A character string of your path file to where you want your output saved
metadata_path	A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to model_path.
ModelID	A character string to name your model and output
ReturnFactorLevels	TRUE or FALSE. Set to FALSE to not return factor levels.
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
Verbose	Set to 0 if you want to suppress model evaluation updates in training
NumOfParDepPlots	Tell the function the number of partial dependence calibration plots you want to create.
NThreads	Set the maximum number of threads you'd like to dedicate to the model run. E.g. 8
eval_metric	This is the metric used to identify best grid tuned model. Choose from "logloss","error","aucpr","auc"
TreeMethod	Choose from "hist", "gpu_hist"
GridTune	Set to TRUE to run a grid tuning procedure
BaselineComparison	Set to either "default" or "best". Default is to compare each successive model build to the baseline model using max trees (from function args). Best makes the comparison to the current best model.

MaxModelsInGrid
: Number of models to test from grid options.

MaxRunsWithoutNewWinner
: A number

MaxRunMinutes
: In minutes

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

Shuffles
: Numeric. List a number to let the program know how many times you want to shuffle the grids for grid tuning

Trees
: Bandit grid partitioned. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the trees numbers you want to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(1000L, 10000L, 1000L)

eta
: Bandit grid partitioned. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the LearningRate values to test. For running grid tuning, a NULL value supplied will mean these values are tested c(0.01,0.02,0.03,0.04)

max_depth
: Bandit grid partitioned. Number, or vector for depth to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(4L, 16L, 2L)

min_child_weight
: Number, or vector for min_child_weight to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(1.0, 10.0, 1.0)

subsample
: Number, or vector for subsample to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(0.55, 1.0, 0.05)

colsample_bytree
: Number, or vector for colsample_bytree to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(0.55, 1.0, 0.05)

Value

Saves to file and returned in list: VariableImportance.csv, Model, ValidationData.csv, Evalution-Plot.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 Automated Supervised Learning - Binary Classification: AutoCatBoostClassifier(), AutoH2oDRFClassifier(), AutoH2oGAMClassifier(), AutoH2oGBMClassifier(), AutoH2oGLMClassifier(), AutoH2oMLClassifier()

Examples

```
## Not run:
# Create some dummy correlated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000L,
  ID = 2L,
  ZIP = 0L,
```

```
  AddDate = FALSE,
  Classification = TRUE,
  MultiClass = FALSE)

# Run function
TestModel <- RemixAutoML::AutoXGBoostClassifier(

    # GPU or CPU
    TreeMethod = "hist",
    NThreads = parallel::detectCores(),

    # Metadata arguments
    model_path = normalizePath("./"),
    metadata_path = NULL,
    ModelID = "Test_Model_1",
    ReturnFactorLevels = TRUE,
    ReturnModelObjects = TRUE,
    SaveModelObjects = FALSE,

    # Data arguments
    data = data,
    TrainOnFull = FALSE,
    ValidationData = NULL,
    TestData = NULL,
    TargetColumnName = "Adrian",
    FeatureColNames = names(data)[!names(data) %chin%
      c("IDcol_1", "IDcol_2","Adrian")],
    IDcols = c("IDcol_1","IDcol_2"),

    # Model evaluation
    eval_metric = "auc",
    NumOfParDepPlots = 3L,

    # Grid tuning arguments
    PassInGrid = NULL,
    GridTune = TRUE,
    BaselineComparison = "default",
    MaxModelsInGrid = 10L,
    MaxRunsWithoutNewWinner = 20L,
    MaxRunMinutes = 24L*60L,
    Verbose = 1L,

    # Trees, Depth, and LearningRate used in the bandit grid tuning
    # Must set Trees to a single value if you are not grid tuning
    # The ones below can be set to NULL and the values in the
    #    example will be used
    Shuffles = 1L,
    Trees = seq(50L, 500L, 50L),
    eta = seq(0.05,0.40,0.05),
    max_depth = seq(4L, 16L, 2L),
    min_child_weight = seq(1.0, 10.0, 1.0),
    subsample = seq(0.55, 1.0, 0.05),
    colsample_bytree = seq(0.55, 1.0, 0.05))

## End(Not run)
```

AutoXGBoostHurdleModel
AutoXGBoostHurdleModel is generalized hurdle modeling framework

Description

AutoXGBoostHurdleModel is generalized hurdle modeling framework

Usage

```
AutoXGBoostHurdleModel(
  TreeMethod = "hist",
  TrainOnFull = FALSE,
  PassInGrid = NULL,
  NThreads = max(1L, parallel::detectCores() - 2L),
  ModelID = "ModelTest",
  Paths = NULL,
  MetaDataPaths = NULL,
  data,
  ValidationData = NULL,
  TestData = NULL,
  Buckets = 0L,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  IDcols = NULL,
  TransformNumericColumns = NULL,
  SplitRatios = c(0.7, 0.2, 0.1),
  SaveModelObjects = FALSE,
  ReturnModelObjects = TRUE,
  NumOfParDepPlots = 10L,
  GridTune = FALSE,
  grid_eval_metric = "accuracy",
  MaxModelsInGrid = 1L,
  BaselineComparison = "default",
  MaxRunsWithoutNewWinner = 10L,
  MaxRunMinutes = 60L,
  Trees = list(classifier = seq(1000, 2000, 100), regression = seq(1000, 2000, 100)),
  eta = list(classifier = seq(0.05, 0.4, 0.05), regression = seq(0.05, 0.4, 0.05)),
  max_depth = list(classifier = seq(4L, 16L, 2L), regression = seq(4L, 16L, 2L)),
  min_child_weight = list(classifier = seq(1, 10, 1), regression = seq(1, 10, 1)),
  subsample = list(classifier = seq(0.55, 1, 0.05), regression = seq(0.55, 1, 0.05)),
  colsample_bytree = list(classifier = seq(0.55, 1, 0.05), regression = seq(0.55, 1,
    0.05))
)
```

Arguments

TreeMethod	Set to hist or gpu_hist depending on if you have an xgboost installation capable of gpu processing
TrainOnFull	Set to TRUE to train model on 100 percent of data

PassInGrid Pass in a grid for changing up the parameter settings for catboost

NThreads Set to the number of threads you would like to dedicate to training

ModelID Define a character name for your models

Paths The path to your folder where you want your model information saved

MetaDataPaths A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to Paths.

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 Primary-DateColumn)

IDcols Includes PrimaryDateColumn and any other columns you want returned in the validation data with predictions

TransformNumericColumns Transform numeric column inside the AutoCatBoostRegression() function

SplitRatios Supply vector of partition ratios. For example, c(0.70,0.20,0,10).

SaveModelObjects Set to TRUE to save the model objects to file in the folders listed in Paths

ReturnModelObjects Set to TRUE to return all model objects

NumOfParDepPlots Set to pull back N number of partial dependence calibration plots.

GridTune Set to TRUE if you want to grid tune the models

grid_eval_metric Select the metric to optimize in grid tuning. "accuracy", "microauc", "logloss"

MaxModelsInGrid Set to a numeric value for the number of models to try in grid tune

BaselineComparison "default"

MaxRunsWithoutNewWinner Number of runs without a new winner before stopping the grid tuning

MaxRunMinutes Max number of minutes to allow the grid tuning to run for

Trees Provide a named list to have different number of trees for each model. Trees = list("classifier" = seq(1000,2000,100), "regression" = seq(1000,2000,100))

eta Provide a named list to have different number of eta for each model.

max_depth Provide a named list to have different number of max_depth for each model.

min_child_weight
: Provide a named list to have different number of min_child_weight for each model.

subsample
: Provide a named list to have different number of subsample for each model.

colsample_bytree
: Provide a named list to have different number of colsample_bytree for each model.

Value

Returns AutoXGBoostRegression() 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 the grid used

Author(s)

Adrian Antico

See Also

Other Supervised Learning - Compound: AutoCatBoostHurdleModel(), AutoCatBoostSizeFreqDist(), AutoH2oDRFHurdleModel(), AutoH2oGBMHurdleModel(), AutoH2oGBMSizeFreqDist()

Examples

```
## Not run:
Output <- RemixAutoML::AutoXGBoostHurdleModel(

   # Operationalization args
   TreeMethod = "hist",
   TrainOnFull = FALSE,
   PassInGrid = NULL,

   # Metadata args
   NThreads = max(1L, parallel::detectCores()-2L),
   ModelID = "ModelTest",
   Paths = normalizePath("./"),
   MetaDataPaths = NULL,

   # data args
   data,
   ValidationData = NULL,
   TestData = NULL,
   Buckets = 0L,
   TargetColumnName = NULL,
   FeatureColNames = NULL,
   IDcols = NULL,

   # options
   TransformNumericColumns = NULL,
   SplitRatios = c(0.70, 0.20, 0.10),
   ReturnModelObjects = TRUE,
   SaveModelObjects = FALSE,
   NumOfParDepPlots = 10L,
```

```
    # grid tuning args
    GridTune = FALSE,
    grid_eval_metric = "accuracy",
    MaxModelsInGrid = 1L,
    BaselineComparison = "default",
    MaxRunsWithoutNewWinner = 10L,
    MaxRunMinutes = 60L,

    # bandit hyperparameters
    Trees = list("classifier" = seq(1000,2000,100),
                 "regression" = seq(1000,2000,100)),
    eta = list("classifier" = seq(0.05,0.40,0.05),
               "regression" = seq(0.05,0.40,0.05)),
    max_depth = list("classifier" = seq(4L,16L,2L),
                     "regression" = seq(4L,16L,2L)),

    # random hyperparameters
    min_child_weight = list("classifier" = seq(1.0,10.0,1.0),
                            "regression" = seq(1.0,10.0,1.0)),
    subsample = list("classifier" = seq(0.55,1.0,0.05),
                     "regression" = seq(0.55,1.0,0.05)),
    colsample_bytree = list("classifier" = seq(0.55,1.0,0.05),
                            "regression" = seq(0.55,1.0,0.05)))

## End(Not run)
```

`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,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  IDcols = NULL,
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  Objective = "multi:softmax",
```

```
  ReturnFactorLevels = TRUE,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  Verbose = 0L,
  NumOfParDepPlots = 3L,
  NThreads = parallel::detectCores(),
  eval_metric = "merror",
  grid_eval_metric = "accuracy",
  TreeMethod = "hist",
  GridTune = FALSE,
  BaselineComparison = "default",
  MaxModelsInGrid = 10L,
  MaxRunsWithoutNewWinner = 20L,
  MaxRunMinutes = 24L * 60L,
  PassInGrid = NULL,
  Shuffles = 1L,
  Trees = 50L,
  eta = NULL,
  max_depth = NULL,
  min_child_weight = NULL,
  subsample = NULL,
  colsample_bytree = NULL
)
```

Arguments

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

`TrainOnFull` Set to TRUE to train on full data

`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.

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

`metadata_path` A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to model_path.

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

`Objective` 'multi:softmax'

`ReturnFactorLevels` TRUE or FALSE. Set to FALSE to not return factor levels.

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

Verbose
: Set to 0 if you want to suppress model evaluation updates in training

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

NThreads
: Set the maximum number of threads you'd like to dedicate to the model run. E.g. 8

eval_metric
: This is the metric used to identify best grid tuned model. Choose from "logloss","error","aucpr","auc"

grid_eval_metric
: "accuracy", "logloss", "microauc"

TreeMethod
: Choose from "hist", "gpu_hist"

GridTune
: Set to TRUE to run a grid tuning procedure

BaselineComparison
: Set to either "default" or "best". Default is to compare each successive model build to the baseline model using max trees (from function args). Best makes the comparison to the current best model.

MaxModelsInGrid
: Number of models to test from grid options.

MaxRunsWithoutNewWinner
: A number

MaxRunMinutes
: In minutes

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

Shuffles
: Numeric. List a number to let the program know how many times you want to shuffle the grids for grid tuning

Trees
: Bandit grid partitioned. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the trees numbers you want to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(1000L, 10000L, 1000L)

eta
: Bandit grid partitioned. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the LearningRate values to test. For running grid tuning, a NULL value supplied will mean these values are tested c(0.01,0.02,0.03,0.04)

max_depth
: Bandit grid partitioned. Number, or vector for depth to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(4L, 16L, 2L)

min_child_weight
: Number, or vector for min_child_weight to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(1.0, 10.0, 1.0)

subsample
: Number, or vector for subsample to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(0.55, 1.0, 0.05)

colsample_bytree
: Number, or vector for colsample_bytree to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(0.55, 1.0, 0.05)

Value

Saves to file and returned in list: VariableImportance.csv, Model, ValidationData.csv, Evaluation-Metrics.csv, GridCollect, GridList, and TargetLevels

Author(s)

Adrian Antico

See Also

Other Automated Supervised Learning - Multiclass Classification: AutoCatBoostMultiClass(), AutoH2oDRFMultiClass(), AutoH2oGAMMultiClass(), AutoH2oGBMMultiClass(), AutoH2oGLMMultiClass(), AutoH2oMLMultiClass()

Examples

```
## Not run:
# Create some dummy correlated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000L,
  ID = 2L,
  ZIP = 0L,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = TRUE)

# Run function
TestModel <- RemixAutoML::AutoXGBoostMultiClass(

    # GPU or CPU
    TreeMethod = "hist",
    NThreads = parallel::detectCores(),

    # Metadata arguments
    model_path = normalizePath("./"),
    metadata_path = file.path(normalizePath("./"),
      "R_Model_Testing"),
    ModelID = "Test_Model_1",
    ReturnFactorLevels = TRUE,
    ReturnModelObjects = TRUE,
    SaveModelObjects = FALSE,

    # Data arguments
    data = data,
    TrainOnFull = FALSE,
    ValidationData = NULL,
    TestData = NULL,
    TargetColumnName = "Adrian",
    FeatureColNames = names(data)[!names(data) %chin%
      c("IDcol_1", "IDcol_2","Adrian")],
    IDcols = c("IDcol_1","IDcol_2"),

    # Model evaluation
    eval_metric = "auc",
    Objective = 'multi:softmax',
```

```
    grid_eval_metric = "accuracy",
    NumOfParDepPlots = 3L,

    # Grid tuning arguments
    PassInGrid = NULL,
    GridTune = TRUE,
    BaselineComparison = "default",
    MaxModelsInGrid = 10L,
    MaxRunsWithoutNewWinner = 20L,
    MaxRunMinutes = 24L*60L,
    Verbose = 1L,

    # Trees, Depth, and LearningRate used in the bandit grid tuning
    # Must set Trees to a single value if you are not grid tuning
    # The ones below can be set to NULL
    #   and the values in the example will be used
    Shuffles = 1L,
    Trees = seq(50L, 500L, 50L),
    eta = seq(0.05,0.40,0.05),
    max_depth = seq(4L, 16L, 2L),
    min_child_weight = seq(1.0, 10.0, 1.0),
    subsample = seq(0.55, 1.0, 0.05),
    colsample_bytree = seq(0.55, 1.0, 0.05))

## End(Not run)
```

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,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  IDcols = NULL,
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
```

```
    ReturnFactorLevels = TRUE,
    ReturnModelObjects = TRUE,
    SaveModelObjects = FALSE,
    TransformNumericColumns = NULL,
   Methods = c("BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson"),
    Verbose = 0L,
    NumOfParDepPlots = 3L,
    NThreads = parallel::detectCores(),
    LossFunction = "reg:squarederror",
    eval_metric = "rmse",
    TreeMethod = "hist",
    GridTune = FALSE,
    grid_eval_metric = "rmse",
    BaselineComparison = "default",
    MaxModelsInGrid = 10L,
    MaxRunsWithoutNewWinner = 20L,
    MaxRunMinutes = 24L * 60L,
    PassInGrid = NULL,
    Shuffles = 1L,
    Trees = 50L,
    eta = NULL,
    max_depth = NULL,
    min_child_weight = NULL,
    subsample = NULL,
    colsample_bytree = NULL
)
```

Arguments

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

`TrainOnFull` Set to TRUE to train on full data

`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.

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

`metadata_path` A character string of your path file to where you want your model evaluation output saved. If left NULL, all output will be saved to model_path.

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

`ReturnFactorLevels` Set to TRUE to have the factor levels returned with the other model objects

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

TransformNumericColumns
: Set to NULL to do nothing; otherwise supply the column names of numeric variables you want transformed

Methods
: Choose from "BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "Yeo-Johnson". Function will determine if one cannot be used because of the underlying data.

Verbose
: Set to 0 if you want to suppress model evaluation updates in training

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

NThreads
: Set the maximum number of threads you'd like to dedicate to the model run. E.g. 8

LossFunction
: Default is 'reg:squarederror'. Other options include 'reg:squaredlogerror', 'reg:pseudohubererror', 'count:poisson', 'survival:cox', 'survival:aft', 'aft_loss_distribution', 'reg:gamma', 'reg:tweedie'

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

TreeMethod
: Choose from "hist", "gpu_hist"

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"

BaselineComparison
: Set to either "default" or "best". Default is to compare each successive model build to the baseline model using max trees (from function args). Best makes the comparison to the current best model.

MaxModelsInGrid
: Number of models to test from grid options (243 total possible options)

MaxRunsWithoutNewWinner
: A number

MaxRunMinutes
: In minutes

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

Shuffles
: Numeric. List a number to let the program know how many times you want to shuffle the grids for grid tuning

Trees
: Bandit grid partitioned. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the trees numbers you want to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(1000L, 10000L, 1000L)

eta
: Bandit grid partitioned. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the LearningRate values to test. For running grid tuning, a NULL value supplied will mean these values are tested c(0.01,0.02,0.03,0.04)

max_depth
: Bandit grid partitioned. Number, or vector for depth to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(4L, 16L, 2L)

min_child_weight
: Number, or vector for min_child_weight to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(1.0, 10.0, 1.0)

subsample
: Number, or vector for subsample to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(0.55, 1.0, 0.05)

colsample_bytree
: Number, or vector for colsample_bytree to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(0.55, 1.0, 0.05)

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 Automated Supervised Learning - Regression: AutoCatBoostRegression(), AutoH2oDRFRegression(), AutoH2oGAMRegression(), AutoH2oGBMRegression(), AutoH2oGLMRegression(), AutoH2oMLRegression()

Examples

```
## Not run:
# Create some dummy correlated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000,
  ID = 2,
  ZIP = 0,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run function
TestModel <- RemixAutoML::AutoXGBoostRegression(

    # GPU or CPU
    TreeMethod = "hist",
    NThreads = NThreads = parallel::detectCores(),
    LossFunction = 'reg:squarederror',

    # Metadata arguments:
    #   'ModelID' is used to create part of the file
    #       names generated when saving to file'
    #   'model_path' is where the minimal model objects
    #       for scoring will be stored
    #   'ModelID' will be the name of the saved model object
    #   'metadata_path' is where model evaluation and model
    #       interpretation files are saved
    #    objects saved to model_path if metadata_path is null
    #    Saved objects include:
    #    'ModelID_ValidationData.csv' is the supplied or generated
```

```
#       TestData with predicted values
#    'ModelID_ROC_Plot.png' and 'Model_ID_EvaluationPlot.png'
#        calibration plot
#    'ModelID_VariableImportance.csv' is the variable importance.
#        This won't be saved to file if GrowPolicy is either
#          "Depthwise" or "Lossguide" was used
#    'ModelID_ExperimentGrid.csv' if GridTune = TRUE.
#        Results of all model builds including parameter settings,
#          bandit probs, and grid IDs
#    'ModelID_EvaluationMetrics.csv' which contains all confusion
#           matrix measures across all thresholds
model_path = normalizePath("./"),
metadata_path = NULL,
ModelID = "Test_Model_1",
ReturnFactorLevels = TRUE,
ReturnModelObjects = TRUE,
SaveModelObjects = FALSE,

# Data arguments:
#   'TrainOnFull' is to train a model with 100 percent of
#      your data.
#   That means no holdout data will be used for evaluation
#   If ValidationData and TestData are NULL and TrainOnFull
#      is FALSE then data will be split 70 20 10
#   'PrimaryDateColumn' is a date column in data that is
#      meaningful when sorted.
#    CatBoost categorical treatment is enhanced when supplied
#   'IDcols' are columns in your data that you don't use for
#      modeling but get returned with ValidationData
data = data,
TrainOnFull = FALSE,
ValidationData = NULL,
TestData = NULL,
TargetColumnName = "Adrian",
FeatureColNames = names(data)[!names(data) %chin%
  c("IDcol_1", "IDcol_2","Adrian")],
IDcols = c("IDcol_1","IDcol_2"),
TransformNumericColumns = NULL,
Methods = c("BoxCox", "Asinh", "Asin", "Log",
  "LogPlus1", "Logit", "YeoJohnson"),

# Model evaluation
eval_metric = "rmse",
NumOfParDepPlots = 3L,

# Grid tuning arguments
PassInGrid = NULL,
GridTune = TRUE,
grid_eval_metric = "mse",
BaselineComparison = "default",
MaxModelsInGrid = 10L,
MaxRunsWithoutNewWinner = 20L,
MaxRunMinutes = 24L*60L,
Verbose = 1L,

# Trees, Depth, and LearningRate used in the bandit grid tuning
# Must set Trees to a single value if you are not grid tuning
```

```
    # The ones below can be set to NULL
    Shuffles = 1L,
    Trees = seq(50L, 500L, 50L),
    eta = seq(0.05,0.40,0.05),
    max_depth = seq(4L, 16L, 2L),
    min_child_weight = seq(1.0, 10.0, 1.0),
    subsample = seq(0.55, 1.0, 0.05),
    colsample_bytree = seq(0.55, 1.0, 0.05))

## End(Not run)
```

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,
  FactorLevelsList = NULL,
  TargetLevels = NULL,
  Objective = "multi:softmax",
  OneHot = FALSE,
  ModelObject = NULL,
  ModelPath = NULL,
  ModelID = NULL,
  ReturnFeatures = TRUE,
  TransformNumeric = FALSE,
  BackTransNumeric = FALSE,
  TargetColumnName = NULL,
  TransformationObject = NULL,
  TransID = NULL,
  TransPath = NULL,
  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

`FactorLevelsList` Supply the factor variables' list from DummifyDT()

`TargetLevels` Supply the target levels output from AutoXGBoostMultiClass() or the scoring function will go looking for it in the file path you supply.

`Objective` Set to 'multi:softprobs' if you did so in training. Default is softmax

`OneHot` Set to TRUE to have one-hot-encoding run. Otherwise, N columns will be made for N levels of a factor variable

`ModelObject` Supply a model for scoring, otherwise it will have to search for it in the file path you specify

`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.

`TransformNumeric` Set to TRUE if you have features that were transformed automatically from an Auto__Regression() model AND you haven't already transformed them.

`BackTransNumeric` Set to TRUE to generate back-transformed predicted values. Also, if you return features, those will also be back-transformed.

`TargetColumnName` Input your target column name used in training if you are utilizing the transformation service

`TransformationObject` Set to NULL if you didn't use transformations or if you want the function to pull from the file output from the Auto__Regression() function. You can also supply the transformation data.table object with the transformation details versus having it pulled from file.

`TransID` Set to the ID used for saving the transformation data.table object or set it to the ModelID if you are pulling from file from a build with Auto__Regression().

`TransPath` Set the path file to the folder where your transformation data.table detail object is stored. If you used the Auto__Regression() to build, set it to the same path as ModelPath.

`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.

Author(s)

Adrian Antico

See Also

Other Automated Model Scoring: AutoCatBoostScoring(), AutoH2OMLScoring(), AutoH2OModeler(), AutoHurdleScoring(), IntermittentDemandScoringDataGenerator()

Examples

```
## Not run:
Preds <- AutoXGBoostScoring(
  TargetType = "regression",
  ScoringData = data,
  FeatureColumnNames = 2:12,
  IDcols = NULL,
  FactorLevelsList = NULL,
  TargetLevels = NULL,
  Objective = "multi:softmax",
  OneHot = FALSE,
  ModelObject = NULL,
  ModelPath = "home",
  ModelID = "ModelTest",
  ReturnFeatures = TRUE,
  TransformNumeric = FALSE,
  BackTransNumeric = FALSE,
  TargetColumnName = NULL,
  TransformationObject = NULL,
  TransID = NULL,
  TransPath = NULL,
  MDP_Impute = TRUE,
  MDP_CharToFactor = TRUE,
  MDP_RemoveDates = TRUE,
  MDP_MissFactor = "0",
  MDP_MissNum = -1)

## End(Not run)
```

CarmaCatBoostKeepVarsGDL *CarmaCatBoostKeepVarsGDL*

Description

CarmaCatBoostKeepVarsGDL is to help manage carma code

Usage

```
CarmaCatBoostKeepVarsGDL(
  data,
  IndepVarPassTRUE = "GroupVar",
  UpdateData,
  CalendarFeatures,
  XREGS,
  Difference,
  HierarchGroups,
  GroupVariables,
  GroupVarVector,
  CalendarVariables,
  HolidayVariable,
  TargetColumnName,
  DateColumnName,
  Preds
)
```

Arguments

`data`	Supply data
`IndepVarPassTRUE`	Name of the column used as a single grouping variable.
`UpdateData`	Supply UpdateData
`CalendarFeatures`	Supply CalendarFeatures
`XREGS`	Supply XREGS
`Difference`	Supply Difference
`HierarchGroups`	Supply HierarchGroups
`GroupVariables`	Supply GroupVariables
`GroupVarVector`	Supply GroupVarVector
`CalendarVariables`	Supply CalendarVariables
`HolidayVariable`	Supply HolidayVariable
`TargetColumnName`	Supply TargetColumnName
`DateColumnName`	Supply DateColumnName
`Preds`	Supply Preds

Author(s)

Adrian Antico

See Also

Other Carma Helper: CARMA_Define_Args(), CARMA_Get_IndepentVariablesPass(), CARMA_GroupHierarchyCheck
CarmaH2OKeepVarsGDL(), CarmaXGBoostKeepVarsGDL()

CarmaH2OKeepVarsGDL   *CarmaH2OKeepVarsGDL*

Description

CarmaH2OKeepVarsGDL is to help manage carma code

Usage

```
CarmaH2OKeepVarsGDL(
  data,
  IndepVarPassTRUE = "GroupVar",
  UpdateData,
  CalendarFeatures,
  XREGS,
  Difference,
  HierarchGroups,
  GroupVariables,
  GroupVarVector,
  CalendarVariables = NULL,
  HolidayVariable = NULL,
  TargetColumnName,
  DateColumnName
)
```

Arguments

`data`	Supply data
`IndepVarPassTRUE`	Name of the column used as a single grouping variable.
`UpdateData`	Supply UpdateData
`CalendarFeatures`	Supply CalendarFeatures
`XREGS`	Supply XREGS
`Difference`	Supply Difference
`HierarchGroups`	Supply HierarchGroups
`GroupVariables`	Supply GroupVariables
`GroupVarVector`	Supply GroupVarVector
`CalendarVariables`	Supply CalendarVariables

HolidayVariable
Supply HolidayVariable

TargetColumnName
Supply TargetColumnName

DateColumnName Supply DateColumnName

Author(s)

Adrian Antico

See Also

Other Carma Helper: CARMA_Define_Args(), CARMA_Get_IndepentVariablesPass(), CARMA_GroupHierarchyCheck
CarmaCatBoostKeepVarsGDL(), CarmaXGBoostKeepVarsGDL()

CarmaHoldoutMetrics *CarmaHoldoutMetrics*

Description

CarmaHoldoutMetrics

Usage

```
CarmaHoldoutMetrics(
  DATA = TestDataEval,
  TARGETCOLUMNNAME = TargetColumnName,
  GROUPVARIABLES = GroupingVariables
)
```

Arguments

DATA TestDataEval

TARGETCOLUMNNAME
TargetColumnName

GROUPVARIABLES GroupVariables

Author(s)

Adrian Antico

See Also

Other Time Series: DifferenceDataReverse(), DifferenceData()

CarmaXGBoostKeepVarsGDL
CarmaXGBoostKeepVarsGDL

Description

CarmaXGBoostKeepVarsGDL is to help manage carma code

Usage

```
CarmaXGBoostKeepVarsGDL(
  data,
  IndepVarPassTRUE = "GroupVar",
  UpdateData,
  CalendarFeatures,
  XREGS,
  Difference,
  HierarchGroups,
  GroupVariables,
  GroupVarVector,
  CalendarVariables = NULL,
  HolidayVariable = NULL,
  TargetColumnName,
  DateColumnName
)
```

Arguments

`data`	Supply data
`IndepVarPassTRUE`	Name of the column used as a single grouping variable.
`UpdateData`	Supply UpdateData
`CalendarFeatures`	Supply CalendarFeatures
`XREGS`	Supply XREGS
`Difference`	Supply Difference
`HierarchGroups`	Supply HierarchGroups
`GroupVariables`	Supply GroupVariables
`GroupVarVector`	Supply GroupVarVector
`CalendarVariables`	Supply CalendarVariables
`HolidayVariable`	Supply HolidayVariable
`TargetColumnName`	Supply TargetColumnName
`DateColumnName`	Supply DateColumnName

Author(s)

Adrian Antico

See Also

Other Carma Helper: CARMA_Define_Args(), CARMA_Get_IndepentVariablesPass(), CARMA_GroupHierarchyCheck
CarmaCatBoostKeepVarsGDL(), CarmaH2OKeepVarsGDL()

CARMA_Define_Args *CARMA_Define_Args*

Description

CARMA_Define_Args is to help manage carma code

Usage

```
CARMA_Define_Args(
  TimeUnit = NULL,
  TimeGroups = NULL,
  HierarchGroups = NULL,
  GroupVariables = NULL,
  FC_Periods = NULL,
  PartitionType = NULL,
  TrainOnFull = NULL,
  SplitRatios = NULL,
  SD_Periods = 0L,
  Skew_Periods = 0L,
  Kurt_Periods = 0L,
  Quantile_Periods = 0L
)
```

Arguments

TimeUnit	= TimeUnit
TimeGroups	= TimeGroups
HierarchGroups	= HierarchGroups
GroupVariables	= GroupVariables
FC_Periods	= FC_Periods
PartitionType	= PartitionType
TrainOnFull	= TrainOnFull
SplitRatios	= SplitRatios
SD_Periods	= 0L turns it off, otherwise values must be greater than 1 such as c(2L,5L,6L,25L)
Skew_Periods	= 0L turns it off, otherwise values must be greater than 2 such as c(3L,5L,6L,25L)
Kurt_Periods	= 0L turns it off, otherwise values must be greater than 3 such as c(4L,5L,6L,25L)
Quantile_Periods	= 0L turns it off, otherwise values must be greater than 3 such as c(5L,6L,25L)

Author(s)

Adrian Antico

See Also

Other Carma Helper: CARMA_Get_IndepentVariablesPass(), CARMA_GroupHierarchyCheck(), CarmaCatBoostKeepVarsGDL(), CarmaH2OKeepVarsGDL(), CarmaXGBoostKeepVarsGDL()

CARMA_Get_IndepentVariablesPass

CARMA_Get_IndepentVariablesPass

Description

CARMA_Get_IndepentVariablesPass is to help manage carma code

Usage

```
CARMA_Get_IndepentVariablesPass(HierarchGroups)
```

Arguments

HierarchGroups	Supply HierarchGroups

Author(s)

Adrian Antico

See Also

Other Carma Helper: CARMA_Define_Args(), CARMA_GroupHierarchyCheck(), CarmaCatBoostKeepVarsGDL(), CarmaH2OKeepVarsGDL(), CarmaXGBoostKeepVarsGDL()

CARMA_GroupHierarchyCheck

CARMA_GroupHierarchyCheck

Description

CARMA_GroupHierarchyCheck

Usage

```
CARMA_GroupHierarchyCheck(
  data = data,
  Group_Variables = GroupVariables,
  HierarchyGroups = HierarchGroups
)
```

Arguments

`data`	data fed into function
`Group_Variables`	Takes GroupVariables from caram function
`HierarchyGroups`	Vector of group variables

Author(s)

Adrian Antico

See Also

Other Carma Helper: `CARMA_Define_Args()`, `CARMA_Get_IndepentVariablesPass()`, `CarmaCatBoostKeepVarsGDL()`, `CarmaH2OKeepVarsGDL()`, `CarmaXGBoostKeepVarsGDL()`

`CatBoostClassifierParams`   *CatBoostClassifierParams*

Description

CatBoostClassifierParams

Usage

```
CatBoostClassifierParams(
  counter = NULL,
  BanditArmsN = NULL,
  HasTime = NULL,
  MetricPeriods = NULL,
  ClassWeights = NULL,
  eval_metric = NULL,
  LossFunction = NULL,
  task_type = NULL,
  NumGPUs = NULL,
  model_path = NULL,
  NewGrid = NULL,
  Grid = NULL,
  ExperimentalGrid = NULL,
  GridClusters = NULL
)
```

Arguments

`counter`	Passthrough
`BanditArmsN`	Passthrough
`HasTime`	Passthrough
`MetricPeriods`	Passthrough

`ClassWeights`	Passthrough
`eval_metric`	Passthrough
`LossFunction`	Passthrough
`task_type`	Passthrough
`NumGPUs`	Passthrough
`model_path`	Passthrough
`NewGrid`	Passthrough
`Grid`	Passthrough
`ExperimentalGrid`	Passthrough
`GridClusters`	Passthrough

Author(s)

Adrian Antico

See Also

Other Supervised Learning: `AutoH2OScoring()`, `CatBoostMultiClassParams()`, `CatBoostParameterGrids()`, `CatBoostRegressionParams()`, `XGBoostClassifierParams()`, `XGBoostMultiClassParams()`, `XGBoostParameterGrids()`, `XGBoostRegressionMetrics()`, `XGBoostRegressionParams()`

`CatBoostMultiClassParams` *CatBoostMultiClassParams*

Description

CatBoostMultiClassParams

Usage

```
CatBoostMultiClassParams(
  counter = NULL,
  BanditArmsN = NULL,
  HasTime = NULL,
  MetricPeriods = NULL,
  ClassWeights = NULL,
  eval_metric = NULL,
  loss_function = NULL,
  task_type = NULL,
  model_path = NULL,
  NewGrid = NULL,
  Grid = NULL,
  ExperimentalGrid = NULL,
  GridClusters = NULL
)
```

Arguments

counter	Passthrough
BanditArmsN	Passthrough
HasTime	Passthrough
MetricPeriods	Passthrough
ClassWeights	Passthrough
eval_metric	Passthrough
loss_function	Passthrough
task_type	Passthrough
model_path	Passthrough
NewGrid	Passthrough
Grid	Passthrough
ExperimentalGrid	Passthrough
GridClusters	Passthrough

Author(s)

Adrian Antico

See Also

Other Supervised Learning: AutoH2OScoring(), CatBoostClassifierParams(), CatBoostParameterGrids(), CatBoostRegressionParams(), XGBoostClassifierParams(), XGBoostMultiClassParams(), XGBoostParameterGrids(), XGBoostRegressionMetrics(), XGBoostRegressionParams()

CatBoostParameterGrids

CatBoostParameterGrids

Description

CatBoostParameterGrids https://catboost.ai/docs/concepts/r-training-parameters.html

Usage

```
CatBoostParameterGrids(
  TaskType = "CPU",
  Shuffles = 1L,
  NTrees = seq(1000L, 10000L, 1000L),
  Depth = seq(4L, 16L, 2L),
  LearningRate = c(0.01, 0.02, 0.03, 0.04),
  L2_Leaf_Reg = seq(1, 10, 1),
  RandomStrength = seq(1, 2, 0.1),
  BorderCount = seq(32, 256, 32),
  RSM = c(0.8, 0.85, 0.9, 0.95, 1),
  BootStrapType = c("Bayesian", "Bernoulli", "Poisson", "MVS", "No"),
  GrowPolicy = c("SymmetricTree", "Depthwise", "Lossguide")
)
```

Arguments

TaskType	"GPU" or "CPU"
Shuffles	The number of shuffles you want to apply to each grid
NTrees	seq(1000L, 10000L, 1000L)
Depth	seq(4L, 16L, 2L)
LearningRate	seq(0.01,.10,0.01)
L2_Leaf_Reg	c(1.0:10.0)
RandomStrength	seq(1, 2, 0.1)
BorderCount	seq(32,256,32)
RSM	CPU ONLY, Random subspace method.c(0.80, 0.85, 0.90, 0.95, 1.0)
BootStrapType	c("Bayesian", "Bernoulli", "Poisson", "MVS", "No")
GrowPolicy	c("SymmetricTree", "Depthwise", "Lossguide")

Value

A list containing data.table's with the parameters shuffled and ready to test in the bandit framework

Author(s)

Adrian Antico

See Also

Other Supervised Learning: AutoH2OScoring(), CatBoostClassifierParams(), CatBoostMultiClassParams(), CatBoostRegressionParams(), XGBoostClassifierParams(), XGBoostMultiClassParams(), XGBoostParameterGrids(), XGBoostRegressionMetrics(), XGBoostRegressionParams()

CatBoostRegressionParams *CatBoostRegressionParams*

Description

CatBoostRegressionParams

Usage

```
CatBoostRegressionParams(
  counter = NULL,
  BanditArmsN = NULL,
  HasTime = NULL,
  MetricPeriods = NULL,
  eval_metric = NULL,
  LossFunction = NULL,
  task_type = NULL,
  NumGPUs = NULL,
  model_path = NULL,
  NewGrid = NULL,
```

```
  Grid = NULL,
  ExperimentalGrid = NULL,
  GridClusters = NULL
)
```

Arguments

counter	Passthrough
BanditArmsN	Passthrough
HasTime	Passthrough
MetricPeriods	Passthrough
eval_metric	Passthrough
LossFunction	Passthrough
task_type	Passthrough
NumGPUs	Passthrough
model_path	Passthrough
NewGrid	Passthrough
Grid	Passthrough
ExperimentalGrid	Passthrough
GridClusters	Passthrough

Author(s)

Adrian Antico

See Also

Other Supervised Learning: AutoH2OScoring(), CatBoostClassifierParams(), CatBoostMultiClassParams(), CatBoostParameterGrids(), XGBoostClassifierParams(), XGBoostMultiClassParams(), XGBoostParameterGri XGBoostRegressionMetrics(), XGBoostRegressionParams()

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,
  AngleX = 35,
  AngleY = 0,
  ChartColor = "lightsteelblue1",
  BorderColor = "darkblue",
  TextColor = "darkblue",
  GridColor = "white",
  BackGroundColor = "gray95",
  LegendPosition = "bottom"
)
```

Arguments

Size	The size of the axis labels and title
AngleX	The angle of the x axis labels
AngleY	The angle of the Y axis labels
ChartColor	"lightsteelblue1",
BorderColor	"darkblue",
TextColor	"darkblue",
GridColor	"white",
BackGroundColor	"gray95",
LegendPosition	Where to place legend

Value

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

Author(s)

Adrian Antico

See Also

Other Misc: AutoH2OTextPrepScoring(), PrintToPDF(), RPM_Binomial_Bandit(), tokenizeH2O()

Examples

```
## Not run:
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)

## End(Not run)
```

ClassificationMetrics *ClassificationMetrics*

Description

ClassificationMetrics

Usage

```
ClassificationMetrics(
  TestData,
  Thresholds,
  Target,
  Predict,
  PositiveOutcome,
  NegativeOutcome,
  CostMatrix = c(1, 0, 0, 1)
)
```

Arguments

`TestData`	Test data from your modeling
`Thresholds`	Value
`Target`	Name of your target variable
`Predict`	Name of your predicted value variable
`PositiveOutcome`	The value of the positive outcome level
`NegativeOutcome`	The value of the negative outcome level
`CostMatrix`	c(True Positive Cost, False Negative Cost, False Positive Cost, True Negative Cost)

Author(s)

Adrian Antico

See Also

Other Model Evaluation: `DT_BinaryConfusionMatrix()`, `RemixClassificationMetrics()`

CLForecast *CLForecast*

Description

CLForecast for generating forecasts

Usage

```
CLForecast(
  data,
  OutputFilePath = NULL,
  FC_BaseFunnelMeasure = NULL,
  SegmentName = NULL,
  MaxDateForecasted = NULL,
  MaxCalendarDate = NULL,
  ArgsList = NULL,
  MaxCohortPeriods = NULL
)
```

Arguments

`data`	N
`OutputFilePath`	P
`FC_BaseFunnelMeasure`	d
`SegmentName`	a
`MaxDateForecasted`	S
`MaxCalendarDate`	S
`ArgsList`	A
`MaxCohortPeriods`	T

Value

S

Author(s)

Adrian Antico

See Also

Other Population Dynamics Forecasting: `CLTrainer()`

CLTrainer *CLTrainer*

Description

CLTrainer is a forecasting model for chain ladder style forecasting

Usage

```
CLTrainer(
  data,
  PartitionRatios = c(0.7, 0.2, 0.1),
  BaseFunnelMeasure = NULL,
  ConversionMeasure = NULL,
  ConversionRateMeasure = NULL,
  CohortPeriodsVariable = NULL,
  CalendarDate = NULL,
  CohortDate = NULL,
  TruncateDate = NULL,
  TimeUnit = c("day"),
  CalendarTimeGroups = c("day", "week", "month"),
  CohortTimeGroups = c("day", "week", "month"),
  TransformTargetVariable = TRUE,
  TransformMethods = c("Identity", "YeoJohnson"),
  AnomalyDetection = list(tstat_high = 3, tstat_low = -2),
  Jobs = c("Evaluate", "Train"),
  SaveModelObjects = TRUE,
  ModelID = "Segment_ID",
  ModelPath = NULL,
  MetaDataPath = NULL,
  TaskType = "CPU",
  NumGPUs = 1,
  DT_Threads = max(1L, parallel::detectCores()),
  EvaluationMetric = "RMSE",
  LossFunction = "RMSE",
  NumOfParDepPlots = 1L,
  MetricPeriods = 50L,
  CalendarVariables = c("wday", "mday", "yday", "week", "isoweek", "month", "quarter",
    "year"),
  HolidayGroups = c("USPublicHolidays", "EasterGroup", "ChristmasGroup",
    "OtherEcclesticalFeasts"),
  ImputeRollStats = -0.001,
  CohortHolidayLags = c(1L, 2L, 7L),
  CohortHolidayMovingAverages = c(3L, 7L),
  CalendarHolidayLags = c(1L, 2L, 7L),
  CalendarHolidayMovingAverages = c(3L, 7L),
  CalendarLags = list(day = c(1L, 7L, 21L), week = c(1L, 4L, 52L), month = c(1L, 6L,
    12L)),
  CalendarMovingAverages = list(day = c(1L, 7L, 21L), week = c(1L, 4L, 52L), month =
    c(1L, 6L, 12L)),
  CalendarStandardDeviations = NULL,
```

```
  CalendarSkews = NULL,
  CalendarKurts = NULL,
  CalendarQuantiles = NULL,
  CalendarQuantilesSelected = "q50",
  CohortLags = list(day = c(1L, 7L, 21L), week = c(1L, 4L, 52L), month = c(1L, 6L,
    12L)),
  CohortMovingAverages = list(day = c(1L, 7L, 21L), week = c(1L, 4L, 52L), month =
    c(1L, 6L, 12L)),
  CohortStandardDeviations = NULL,
  CohortSkews = NULL,
  CohortKurts = NULL,
  CohortQuantiles = NULL,
  CohortQuantilesSelected = "q50",
  PassInGrid = NULL,
  GridTune = FALSE,
  BaselineComparison = "default",
  MaxModelsInGrid = 25L,
  MaxRunMinutes = 180L,
  MaxRunsWithoutNewWinner = 10L,
  Trees = 3000L,
  Depth = seq(4L, 8L, 1L),
  LearningRate = seq(0.01, 0.1, 0.01),
  L2_Leaf_Reg = seq(1, 10, 1),
  RSM = c(0.8, 0.85, 0.9, 0.95, 1),
  BootStrapType = c("Bayesian", "Bernoulli", "Poisson", "MVS", "No"),
  GrowPolicy = c("SymmetricTree", "Depthwise", "Lossguide")
)
```

Arguments

`data` data object

`PartitionRatios`
Requires three values for train, validation, and test data sets

`BaseFunnelMeasure`
E.g. "Leads". This value should be a forward looking variable. Say you want to forecast ConversionMeasure 2 months into the future. You should have two months into the future of values of BaseFunnelMeasure

`ConversionMeasure`
E.g. "Conversions". Rate is derived as conversions over leads by cohort periods out

`ConversionRateMeasure`
Conversions over Leads for every cohort

`CohortPeriodsVariable`
Numeric. Numerical value of the the number of periods since cohort base date.

`CalendarDate` The name of your date column that represents the calendar date

`CohortDate` The name of your date column that represents the cohort date

`TruncateDate` NULL. Supply a date to represent the earliest point in time you want in your data. Filtering takes place before partitioning data so feature engineering can include as many non null values as possible.

`TimeUnit` Base time unit of data. "days", "weeks", "months", "quarters", "years"

CalendarTimeGroups
: TimeUnit value must be included. If you want to generate lags and moving averages in several time based aggregations, choose from "days", "weeks", "months", "quarters", "years".

CohortTimeGroups
: TimeUnit value must be included. If you want to generate lags and moving averages in several time based aggregations, choose from "days", "weeks", "months", "quarters", "years".

TransformTargetVariable
: TRUE or FALSe

TransformMethods
: Choose from "Identity", "BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Logit", "YeoJohnson"

AnomalyDetection
: Provide a named list. See examples

Jobs
: Default is "eval" and "train"

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

ModelID
: A character string to name your model and output

ModelPath
: Path to where you want your models saved

MetaDataPath
: Path to where you want your metadata saved. If NULL, function will try ModelPath if it is not NULL.

TaskType
: "GPU" or "CPU" for catboost training

NumGPUs
: Number of GPU's you would like to utilize

DT_Threads
: Number of threads to use for data.table. Default is Total - 2

EvaluationMetric
: 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".

LossFunction
: Used in model training for model fitting. Select from 'RMSE', 'MAE', 'Quantile', 'LogLinQuantile', 'MAPE', 'Poisson', 'PairLogitPairwise', 'Tweedie', 'QueryRMSE'

NumOfParDepPlots
: Number of partial dependence plots to return

MetricPeriods
: Number of trees to build before the internal catboost eval step happens

CalendarVariables
: "wday", "mday", "yday", "week", "isoweek", "month", "quarter", "year"

HolidayGroups
: c("USPublicHolidays","EasterGroup","ChristmasGroup","OtherEcclesticalFeasts")

ImputeRollStats
: Constant value to fill NA after running AutoLagRollStats()

CohortHolidayLags
: c(1L, 2L, 7L),

CohortHolidayMovingAverages
: c(3L, 7L),

CalendarHolidayLags
: c(1L, 2L, 7L),

CalendarHolidayMovingAverages
: = c(3L, 7L),

CalendarLags List of the form list("day" = c(1L, 7L, 21L), "week" = c(1L, 4L, 52L), "month" = c(1L, 6L, 12L))

CalendarMovingAverages
List of the form list("day" = c(1L, 7L, 21L), "week" = c(1L, 4L, 52L), "month" = c(1L, 6L, 12L))

CalendarStandardDeviations
List of the form list("day" = c(1L, 7L, 21L), "week" = c(1L, 4L, 52L), "month" = c(1L, 6L, 12L))

CalendarSkews List of the form list("day" = c(1L, 7L, 21L), "week" = c(1L, 4L, 52L), "month" = c(1L, 6L, 12L))

CalendarKurts List of the form list("day" = c(1L, 7L, 21L), "week" = c(1L, 4L, 52L), "month" = c(1L, 6L, 12L))

CalendarQuantiles
List of the form list("day" = c(1L, 7L, 21L), "week" = c(1L, 4L, 52L), "month" = c(1L, 6L, 12L))

CalendarQuantilesSelected
Supply a vector of "q5", "q10", "q15", "q20", "q25", "q30", "q35", "q40", "q45", "q50", "q55", "q60", "q65", "q70", "q75", "q80", "q85", "q90", "q95"

CohortLags List of the form list("day" = c(1L, 7L, 21L), "week" = c(1L, 4L, 52L), "month" = c(1L, 6L, 12L))

CohortMovingAverages
List of the form list("day" = c(1L, 7L, 21L), "week" = c(1L, 4L, 52L), "month" = c(1L, 6L, 12L))

CohortStandardDeviations
List of the form list("day" = c(1L, 7L, 21L), "week" = c(1L, 4L, 52L), "month" = c(1L, 6L, 12L))

CohortSkews List of the form list("day" = c(1L, 7L, 21L), "week" = c(1L, 4L, 52L), "month" = c(1L, 6L, 12L))

CohortKurts List of the form list("day" = c(1L, 7L, 21L), "week" = c(1L, 4L, 52L), "month" = c(1L, 6L, 12L))

CohortQuantiles
List of the form list("day" = c(1L, 7L, 21L), "week" = c(1L, 4L, 52L), "month" = c(1L, 6L, 12L))

CohortQuantilesSelected
Supply a vector of "q5", "q10", "q15", "q20", "q25", "q30", "q35", "q40", "q45", "q50", "q55", "q60", "q65", "q70", "q75", "q80", "q85", "q90", "q95"

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)

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.

BaselineComparison
Set to either "default" or "best". Default is to compare each successive model build to the baseline model using max trees (from function args). Best makes the comparison to the current best model.

MaxModelsInGrid
Number of models to test from grid options

MaxRunMinutes Maximum number of minutes to let this run

MaxRunsWithoutNewWinner
Number of models built before calling it quits

Trees	Bandit grid partitioned. The maximum number of trees you want in your models
Depth	Bandit grid partitioned. Number, or vector for depth to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(4L, 16L, 2L)
LearningRate	Bandit grid partitioned. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the LearningRate values to test. For running grid tuning, a NULL value supplied will mean these values are tested c(0.01,0.02,0.03,0.04)
L2_Leaf_Reg	Random testing. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the L2_Leaf_Reg values to test. For running grid tuning, a NULL value supplied will mean these values are tested seq(1.0, 10.0, 1.0)
RSM	CPU only. Random testing. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the RSM values to test. For running grid tuning, a NULL value supplied will mean these values are tested c(0.80, 0.85, 0.90, 0.95, 1.0)
BootStrapType	Random testing. Supply a single value for non-grid tuning cases. Otherwise, supply a vector for the BootStrapType values to test. For running grid tuning, a NULL value supplied will mean these values are tested c("Bayesian", "Bernoulli", "Poisson", "MVS", "No")
GrowPolicy	Random testing. NULL, character, or vector for GrowPolicy to test. For grid tuning, supply a vector of values. For running grid tuning, a NULL value supplied will mean these values are tested c("SymmetricTree", "Depthwise", "Lossguide")

Value

Saves metadata and models to files of your choice. Also returns metadata and models from the function. User specifies both options.

Author(s)

Adrian Antico

See Also

Other Population Dynamics Forecasting: CLForecast()

Examples

```
## Not run:
# Create simulated data
data <- RemixAutoML::FakeDataGenerator(
  ChainLadderData = TRUE)

# Build model
RemixAutoML::CLTrainer(

   # Data Arguments----
   data = data,
   PartitionRatios = c(0.70,0.20,0.10),
   BaseFunnelMeasure = "Leads",
   ConversionMeasure = "Appointments",
   ConversionRateMeasure = NULL,
   CohortPeriodsVariable = "CohortDays",
```

```
CalendarDate = "CalendarDateColumn",
CohortDate = "CohortDateColumn",
TruncateDate = NULL,
TimeUnit = "days",
TransformTargetVariable = TRUE,
TransformMethods = c("Identity","BoxCox","Asinh",
                     "Asin","LogPlus1","Logit",
                     "YeoJohnson"),
AnomalyDetection = list(tstat_high = 3,
  tstat_low = -2),

# MetaData Arguments----
Jobs = c("eval","train"),
SaveModelObjects = TRUE,
ModelID = "ModelTest",
ModelPath = getwd(),
MetaDataPath = NULL,
TaskType = "GPU",
NumGPUs = 1,
DT_Threads = max(1L, parallel::detectCores() - 2L),
EvaluationMetric = "RMSE",
LossFunction = "RMSE",
NumOfParDepPlots = 1L,
MetricPeriods = 50L,

# Feature Engineering Arguments----
ImputeRollStats = -0.001,
CalendarTimeGroups = c("days","weeks","months"),
CohortTimeGroups = c("days", "weeks"),
CalendarVariables = c("wday","mday","yday","week",
                      "month","quarter","year"),
HolidayGroups = c("USPublicHolidays","EasterGroup",
                  "ChristmasGroup","OtherEcclesticalFeasts"),
CohortHolidayLags = c(1L,2L,7L),
CohortHolidayMovingAverages = c(3L,7L),
CalendarHolidayLags = c(1L,2L,7L),
CalendarHolidayMovingAverages = c(3L,7L),
CalendarLags = list("day" = c(1L,2L,7L,35L,42L),
                    "week" = c(5L,6L,10L,12L,25L,26L)),
CalendarMovingAverages = list("day" = c(7L,14L,35L,42L),
                              "week" = c(5L,6L,10L,12L,20L,24L),
                              "month" = c(6L,12L)),
CalendarStandardDeviations = NULL,
CalendarSkews = NULL,
CalendarKurts = NULL,
CalendarQuantiles = NULL,
CalendarQuantilesSelected = "q50",
CohortLags = list("day" = c(1L,2L,7L,35L,42L),
                  "week" = c(5L,6L)),
CohortMovingAverages = list("day" = c(7L,14L,35L,42L),
                            "week" = c(5L,6L),
                            "month" = c(1L,2L)),
CohortStandardDeviations = NULL,
CohortSkews = NULL,
CohortKurts = NULL,
CohortQuantiles = NULL,
CohortQuantilesSelected = "q50",
```

```
    # Grid Tuning
    PassInGrid = NULL,
    GridTune = FALSE,
    BaselineComparison = "default",
    MaxModelsInGrid = 25L,
    MaxRunMinutes = 180L,
    MaxRunsWithoutNewWinner = 10L,
    Trees = 1000L,
    Depth = seq(4L,8L,1L),
    LearningRate = seq(0.01,0.10,0.01),
    L2_Leaf_Reg = seq(1.0,10.0,1.0),
    RSM = c(0.80,0.85,0.90,0.95,1.0),
    BootStrapType = c("Bayesian","Bernoulli","Poisson","MVS","No"),
    GrowPolicy = c("SymmetricTree","Depthwise","Lossguide"))

## End(Not run)
```

ColumnSubsetDataTable *ColumnSubsetDataTable*

Description

ColumnSubsetDataTable will subset data tables by column

Usage

```
ColumnSubsetDataTable(
  data,
  TargetColumnName = NULL,
  DateColumnName = NULL,
  GroupVars = NULL
)
```

Arguments

data	data.table
TargetColumnName	Target variable
DateColumnName	Date variable
GroupVars	Group variables

Author(s)

Adrian Antico

See Also

Other Data Wrangling: AutoDataDictionaries(), DataDisplayMeta(), FakeDataGenerator(), FullFactorialCatFeatures(), SQL_ClearTable(), SQL_DropTable(), SQL_Query_Push(), SQL_Query(), SQL_SaveTable(), SQL_Server_DBConnection(), SQL_UpdateTable(), TimeSeriesMelt()

ContinuousTimeDataGenerator

ContinuousTimeDataGenerator for creating continuous time data sets for on demand modeling

Description

ContinuousTimeDataGenerator for creating continuous time data sets for on demand modeling of transactional panel data.

Usage

```
ContinuousTimeDataGenerator(
  data,
  RestrictDateRange = TRUE,
  Case = 2L,
  FC_Periods = 52L,
  SaveData = FALSE,
  FilePath = NULL,
  TargetVariableName = "qty",
  DateVariableName = "date",
  GDL_Targets = NULL,
  TimeUnit = "raw",
  TimeGroups = c("raw", "day", "week"),
  GroupingVariables = "sku",
  HierarchyGroupVars = NULL,
  MinTimeWindow = 1L,
  MinTxnRecords = 2L,
  Lags = 1L:7L,
  MA_Periods = 10L,
  SD_Periods = 10L,
  Skew_Periods = 10L,
  Kurt_Periods = 10L,
  Quantile_Periods = 10L,
  Quantiles_Selected = c("q5"),
  HolidayLags = c(1L:7L),
  HolidayMovingAverages = c(2L:14L),
  TimeBetween = NULL,
  TimeTrendVariable = TRUE,
 CalendarVariables = c("wday", "mday", "yday", "week", "isoweek", "month", "quarter",
    "year"),
  HolidayGroups = "USPublicHolidays",
  PowerRate = 0.5,
  SampleRate = 5,
  TargetWindowSamples = 5,
  PrintSteps = TRUE
)
```

Arguments

data	This is your transactional level data

RestrictDateRange
: Set to TRUE to only pull samples by entity within the entity life (not beyond)

Case
: Currently set as 1 for forecasting and 2 for other

FC_Periods
: The number of future periods to collect data on

SaveData
: Set to TRUE to save the MetaData and final modeling data sets to file

FilePath
: Set to your file of choice for where you want the data sets saved

TargetVariableName
: The name of your target variable that represents demand

DateVariableName
: The date variable of the demand instances

GDL_Targets
: The variable names to run through AutoLagRollStats()

TimeUnit
: List the time unit your data is aggregated by. E.g. "day", "week", "month", "quarter", "year"

TimeGroups
: = c("raw","day","week"),

GroupingVariables
: These variables (or sinlge variable) is the combination of categorical variables that uniquely defines the level of granularity of each individual level to forecast. E.g. "sku" or c("Store","Department"). Sku is typically unique for all sku's. Store and Department in combination defines all unique departments as the department may be repeated across the stores.

HierarchyGroupVars
: Group vars

MinTimeWindow
: The number of time periods you would like to omit for training. Default is 1 so that at a minimum, there is at least one period of values to forecast. You can set it up to a larger value if you do not want more possible target windows for the lower target window values.

MinTxnRecords
: I typically set this to 2 so that there is at least one other instance of demand so that the forecasted values are not complete nonsense.

Lags
: Select the periods for all lag variables you want to create. E.g. c(1:5,52)

MA_Periods
: Select the periods for all moving average variables you want to create. E.g. c(1:5,52)

SD_Periods
: Select the periods for all sd variables you want to create. E.g. c(1:5,52)

Skew_Periods
: Select the periods for all skew variables you want to create. E.g. c(1:5,52)

Kurt_Periods
: Select the periods for all kurtosis variables you want to create. E.g. c(1:5,52)

Quantile_Periods
: Select the periods for all quantiles variables you want to create. E.g. c(1:5,52)

Quantiles_Selected
: Select the quantiles you want. q5, q10, ..., q95

HolidayLags
: Select the lags you want generated

HolidayMovingAverages
: Select the moving averages you want generated

TimeBetween
: Supply a name or NULL

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.

`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

`HolidayGroups`
: Input the holiday groups of your choice from the CreateHolidayVariable() function in this package

`PowerRate`
: Sampling parameter

`SampleRate`
: Set this to a value greater than 0. The calculation used is the number of records per group level raised to the power of PowerRate. Then that values is multiplied by SampleRate.

`TargetWindowSamples`
: = 5

`PrintSteps`
: Set to TRUE to have operation steps printed to the console

Value

Returns two data.table data sets: The first is a modeling data set for the count distribution while the second data set if for the size model data set.

Author(s)

Adrian Antico

See Also

Other Feature Engineering: `AutoDataPartition()`, `AutoHierarchicalFourier()`, `AutoLagRollStatsScoring()`, `AutoLagRollStats()`, `AutoTransformationCreate()`, `AutoTransformationScore()`, `AutoWord2VecModeler()`, `CreateCalendarVariables()`, `CreateHolidayVariables()`, `DT_GDL_Feature_Engineering()`, `DummifyDT()`, `H2oAutoencoder()`, `ModelDataPrep()`, `Partial_DT_GDL_Feature_Engineering()`, `TimeSeriesFill()`

Examples

```
## Not run:
DataSets <- ContinuousTimeDataGenerator(
  data,
  RestrictDateRange = TRUE,
  FC_Periods = 52,
  SaveData = FALSE,
  FilePath = normalizePath("./"),
  TargetVariableName = "qty",
  DateVariableName = "date",
  GDL_Targets = NULL,
  GroupingVariables = "sku",
  HierarchyGroupVars = NULL,
  TimeGroups = c("raw","day","week"),
  MinTimeWindow = 1,
  MinTxnRecords = 2,
  Lags = 1:7,
  MA_Periods = 10L,
  SD_Periods = 10L,
  Skew_Periods = 10L,
  Kurt_Periods = 10L,
  Quantile_Periods = 10L,
```

```
  Quantiles_Selected = c("q5"),
  HolidayLags = c(1L:7L),
  HolidayMovingAverages = c(2L:14L),
  TimeBetween = NULL,
  TimeTrendVariable = TRUE,
  TimeUnit = "day",
  CalendarVariables = c("wday",
    "mday",
    "yday",
    "week",
    "isoweek",
    "month",
    "quarter",
    "year"),
  HolidayGroups = "USPublicHolidays",
  PowerRate = 0.5,
  SampleRate = 5,
  TargetWindowSamples = 5,
  PrintSteps = TRUE)
CountModelData <- DataSets$CountModelData
SizeModelData <- DataSets$SizeModelData
rm(DataSets)

## End(Not run)
```

CreateCalendarVariables

CreateCalendarVariables Create Calendar Variables

Description

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

Usage

```
CreateCalendarVariables(
  data,
  DateCols = NULL,
  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", "wom" (week of month), "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(), AutoHierarchicalFourier(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), ContinuousTimeDataGenerator(), CreateHolidayVariables(), DT_GDL_Feature_Engineering(), DummifyDT(), H2oAutoencoder(), ModelDataPrep(), Partial_DT_GDL_Feature_Engineering(), TimeSeriesFill()

Examples

```
## Not run:
# Create fake data with a Date column----
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.75,
  N = 25000L,
  ID = 2L,
  ZIP = 0L,
  FactorCount = 4L,
  AddDate = TRUE,
  Classification = FALSE,
  MultiClass = FALSE)
for(i in seq_len(20L)) {
  print(i)
  data <- data.table::rbindlist(
    list(data, RemixAutoML::FakeDataGenerator(
    Correlation = 0.75,
    N = 25000L,
    ID = 2L,
    ZIP = 0L,
    FactorCount = 4L,
    AddDate = TRUE,
    Classification = FALSE,
    MultiClass = FALSE)))
}

# Create calendar variables - automatically excludes
#   the second, minute, and hour selections since
#   it is not timestamp data
runtime <- system.time(
  data <- RemixAutoML::CreateCalendarVariables(
    data = data,
    DateCols = "DateTime",
    AsFactor = FALSE,
    TimeUnits = c("second",
                  "minute",
                  "hour",
                  "wday",
                  "mday",
                  "yday",
```

```
                    "week",
                    "isoweek",
                    "wom",
                    "month",
                    "quarter",
                    "year")))
head(data)
print(runtime)

## End(Not run)
```

CreateHolidayVariables

CreateHolidayVariables Create Holiday Count Columns

Description

CreateHolidayVariables Rapidly creates holiday count variables based on the date columns you provide

Usage

```
CreateHolidayVariables(
  data,
  DateCols = NULL,
  HolidayGroups = c("USPublicHolidays", "EasterGroup", "ChristmasGroup",
    "OtherEcclesticalFeasts"),
  Holidays = NULL,
  GroupingVars = NULL,
  Print = FALSE
)
```

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
HolidayGroups	Pick groups
Holidays	Pick holidays
GroupingVars	Grouping variable names
Print	Set to TRUE to print iteration number to console

Value

Returns your data.table with the added holiday indicator variable

Author(s)

Adrian Antico

See Also

Other Feature Engineering: AutoDataPartition(), AutoHierarchicalFourier(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), ContinuousTimeDataGenerator(), CreateCalendarVariables(), DT_GDL_Feature_Engineering(), DummifyDT(), H2oAutoencoder(), ModelDataPrep(), Partial_DT_GDL_Feature_Engineering(), TimeSeriesFill()

Examples

```
## Not run:
# Create fake data with a Date----
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.75,
  N = 25000L,
  ID = 2L,
  ZIP = 0L,
  FactorCount = 4L,
  AddDate = TRUE,
  Classification = FALSE,
  MultiClass = FALSE)
for(i in seq_len(20L)) {
  print(i)
  data <- data.table::rbindlist(list(data,
  RemixAutoML::FakeDataGenerator(
    Correlation = 0.75,
    N = 25000L,
    ID = 2L,
    ZIP = 0L,
    FactorCount = 4L,
    AddDate = TRUE,
    Classification = FALSE,
    MultiClass = FALSE)))
}
# Run function and time it
runtime <- system.time(
  data <- CreateHolidayVariables(
    data,
    DateCols = "DateTime",
    HolidayGroups = c("USPublicHolidays","EasterGroup",
      "ChristmasGroup","OtherEcclesticalFeasts"),
    Holidays = NULL,
    GroupingVars = c("Factor_1","Factor_2","Factor_3","Factor_4"),
    Print = FALSE))
head(data)
print(runtime)

## End(Not run)
```

CreateProjectFolders *CreateProjectFolders Converts path files to proper path files*

Description

CreateProjectFolders Converts path files to proper path files

Usage

```
CreateProjectFolders(
  ProjectName = input$ID_NewProjectName,
  RootPath = input$ID_Root_Folder,
  ExistsButNoProjectList = FALSE,
  Local = FALSE
)
```

Arguments

`ProjectName`	This is the name of a project which will be the name of the file created in the root folder
`RootPath`	This is the path file to the root folder
`ExistsButNoProjectList`	Set to TRUE if the folder exists but not the ProjectList file
`Local`	a

Value

Returns a proper path file string

Author(s)

Adrian Antico

`DataDisplayMeta` *DataDisplayMeta*

Description

DataDisplayMeta

Usage

```
DataDisplayMeta(data)
```

Arguments

`data`	Source data

Author(s)

Adrian Antico

See Also

Other Data Wrangling: `AutoDataDictionaries()`, `ColumnSubsetDataTable()`, `FakeDataGenerator()`, `FullFactorialCatFeatures()`, `SQL_ClearTable()`, `SQL_DropTable()`, `SQL_Query_Push()`, `SQL_Query()`, `SQL_SaveTable()`, `SQL_Server_DBConnection()`, `SQL_UpdateTable()`, `TimeSeriesMelt()`

DifferenceData *DifferenceData*

Description

DifferenceData differences your data set

Usage

```
DifferenceData(
  data,
  ColumnsToDiff = c(names(data)[2:ncol(data)]),
  CARMA = FALSE,
  TargetVariable = NULL,
  GroupingVariable = NULL
)
```

Arguments

`data`	Source data
`ColumnsToDiff`	The column numbers you want differenced
`CARMA`	Set to TRUE for CARMA functions
`TargetVariable`	The target variable name
`GroupingVariable`	Difference data by group

Author(s)

Adrian Antico

See Also

Other Time Series: `CarmaHoldoutMetrics()`, `DifferenceDataReverse()`

DifferenceDataReverse *DifferenceDataReverse*

Description

DifferenceDataReverse reverses the difference

Usage

```
DifferenceDataReverse(
  data,
  ScoreData = Forecasts$Predictions,
  LastRow = DiffTrainOutput$LastRow$Weekly_Sales,
  CARMA = FALSE,
  TargetCol = TargetColumnName,
  FirstRow = DiffTrainOutput$FirstRow,
  GroupingVariables = NULL
)
```

Arguments

`data`	Pre differenced scoring data
`ScoreData`	Predicted values from ML model
`LastRow`	The last row from training data target variables
`CARMA`	Set to TRUE for CARMA utilization
`TargetCol`	Target column name
`FirstRow`	The first row of the target variable
`GroupingVariables`	Group columns

Author(s)

Adrian Antico

See Also

Other Time Series: `CarmaHoldoutMetrics()`, `DifferenceData()`

`DownloadCSVFromStorageExplorer` *DownloadCSVFromStorageExplorer*

Description

DownloadCSVFromStorageExplorer

Usage

```
DownloadCSVFromStorageExplorer(
  UploadCSVObjectName = "data.csv",
  SaveCSVFilePath = file.path(Root),
  SaveCSVName = "RawData.csv",
  UploadLocation = "Analytics Sandbox/Machine Learning",
  DataStoreName = NULL
)
```

Arguments

`UploadCSVObjectName`
: Name of the file you uploaded to the Microsoft Azure Storage Explorer

`SaveCSVFilePath`
: Path file to where you want to save your csv in Azure

`SaveCSVName`
: The name you want to give the csv that will be saved

`UploadLocation`
: The location to where the data is saved in the Azure Storage Explorer

`DataStoreName`
: The name of the store in data factory where you uploaded your data

Author(s)

Adrian Antico

`DT_BinaryConfusionMatrix`

DT_BinaryConfusionMatrix

Description

DT_BinaryConfusionMatrix is for computing all metrics related to binary modeling outcomes

Usage

```
DT_BinaryConfusionMatrix(
  data = MetricsData,
  GroupVariables = "IntervalNum",
  Target = "ActiveAtInterval",
  Predicted = "p1"
)
```

Arguments

`data`
: Supply your model validation data with predictions

`GroupVariables`
: Supply grouping variables to generate statistics by groups

`Target`
: The name of your target variable column

`Predicted`
: The name of your predicted value column#'

Author(s)

Adrian Antico

See Also

Other Model Evaluation: `ClassificationMetrics()`, `RemixClassificationMetrics()`

Examples

```
## Not run:
AggMetricsByGroup <- DT_BinaryConfusionMatrix(
  data,
  GroupVariables = c("Store","Dept"),
  Target = "HitTarget",
  Predicted = "p1")

## End(Not run)
```

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)),
  SDperiods = c(seq(5, 95, 5)),
  Skewperiods = c(seq(5, 95, 5)),
  Kurtperiods = c(seq(5, 95, 5)),
  Quantileperiods = c(seq(5, 95, 5)),
  statsFUNs = c("mean"),
  targets = NULL,
  groupingVars = NULL,
  sortDateName = NULL,
  timeDiffTarget = NULL,
  timeAgg = c("days"),
  WindowingLag = 0,
  Type = c("Lag"),
  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.

`SDperiods`	A numeric vector of Standard Deviation rolling statistics window sizes you want to utilize in the calculations.
`Skewperiods`	A numeric vector of Skewness rolling statistics window sizes you want to utilize in the calculations.
`Kurtperiods`	A numeric vector of Kurtosis rolling statistics window sizes you want to utilize in the calculations.
`Quantileperiods`	A numeric vector of Quantile rolling statistics window sizes you want to utilize in the calculations.
`statsFUNs`	Select from the following c("mean","sd","skew","kurt","q5","q10","q15","q20","q25","q30","q35","q
`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
`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()`, `AutoHierarchicalFourier()`, `AutoLagRollStatsScoring()`, `AutoLagRollStats()`, `AutoTransformationCreate()`, `AutoTransformationScore()`, `AutoWord2VecModeler()`, `ContinuousTimeDataGenerator()`, `CreateCalendarVariables()`, `CreateHolidayVariables()`, `DummifyDT()`, `H2oAutoencoder()`, `ModelDataPrep()`, `Partial_DT_GDL_Feature_Engineering()`, `TimeSeriesFill()`

Examples

```
## Not run:
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),
    SDperiods       = c(seq(5, 95, 5)),
    Skewperiods     = c(seq(5, 95, 5)),
    Kurtperiods     = c(seq(5, 95, 5)),
    Quantileperiods = c(seq(5, 95, 5)),
    statsFUNs      = c("mean",
      "sd","skew","kurt","q05","q95"),
    targets        = c("Target"),
    groupingVars   = NULL,
    sortDateName   = "DateTime",
    timeDiffTarget = c("Time_Gap"),
    timeAgg        = c("days"),
    WindowingLag   = 1,
    Type           = "Lag",
    SimpleImpute   = TRUE)

## End(Not run)
```

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,
  FactorLevelsList = NULL,
  ClustScore = FALSE,
  ReturnFactorLevels = FALSE,
  GroupVar = 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.

FactorLevelsList
Supply a list of factor variable levels

ClustScore This is for scoring AutoKMeans. Set to FALSE for all other applications.

ReturnFactorLevels
If you want a named list of all the factor levels returned, set this to TRUE. Doing so will cause the function to return a list with the source data.table and the list of factor variables' levels

GroupVar Ignore this

Value

Either a data table with new dummy variables columns and optionally removes base columns (if ReturnFactorLevels is FALSE), otherwise a list with the data.table and a list of the factor levels.

Author(s)

Adrian Antico

See Also

Other Feature Engineering: AutoDataPartition(), AutoHierarchicalFourier(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), ContinuousTimeDataGenerator(), CreateCalendarVariables(), CreateHolidayVariables(), DT_GDL_Feature_Engineering(), H2oAutoencoder(), ModelDataPrep(), Partial_DT_GDL_Feature_Engineering TimeSeriesFill()

Examples

```
## Not run:
# Create fake data with 10 categorical columns
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 25000,
  ID = 2L,
  ZIP = 0,
  FactorCount = 10L,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Create dummy variables
data <- RemixAutoML::DummifyDT(
  data = data,
```

```
  cols = c("Factor_1",
           "Factor_2",
           "Factor_3",
           "Factor_4",
           "Factor_5",
           "Factor_6",
           "Factor_8",
           "Factor_9",
           "Factor_10"),
  KeepFactorCols = FALSE,
  OneHot = FALSE,
  SaveFactorLevels = FALSE,
  SavePath = normalizePath("./"),
  ImportFactorLevels = FALSE,
  FactorLevelsList = NULL,
  ClustScore = FALSE,
  ReturnFactorLevels = FALSE)

## End(Not run)
```

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: AutoLimeAid(), LimeModel(), ParDepCalPlots(), RedYellowGreen(), threshOptim()

Examples

```
## Not run:
# Create fake data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.70, N = 10000000, Classification = TRUE)
data.table::setnames(data, "IDcol_1", "Predict")

# Run function
EvalPlot(data,
         PredictionColName = "Predict",
         TargetColName = "Adrian",
         GraphType = "calibration",
         PercentileBucket = 0.05,
         aggrfun = function(x) mean(x, na.rm = TRUE))

## End(Not run)
```

FakeDataGenerator *FakeDataGenerator*

Description

FakeDataGenerator

Usage

```
FakeDataGenerator(
  Correlation = 0.7,
  N = 1000L,
  ID = 5L,
  FactorCount = 2L,
  AddDate = TRUE,
  ZIP = 5L,
  TimeSeries = FALSE,
  TimeSeriesTimeAgg = "day",
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE
)
```

Arguments

`Correlation`	Set the correlation value for simulated data
`N`	Number of records
`ID`	Number of IDcols to include
`FactorCount`	Number of factor type columns to create
`AddDate`	Set to TRUE to include a date column
`ZIP`	Zero Inflation Model target variable creation. Select from 0 to 5 to create that number of distinctly distributed data, stratifed from small to large
`TimeSeries`	For testing AutoBanditSarima
`TimeSeriesTimeAgg`	Choose from "1min", "5min", "10min", "15min", "30min", "hour", "day", "week", "month", "quarter", "year",
`ChainLadderData`	Set to TRUE to return Chain Ladder Data for using AutoMLChainLadderTrainer
`Classification`	Set to TRUE to build classification data
`MultiClass`	Set to TRUE to build MultiClass data

Author(s)

Adrian Antico

See Also

Other Data Wrangling: `AutoDataDictionaries()`, `ColumnSubsetDataTable()`, `DataDisplayMeta()`, `FullFactorialCatFeatures()`, `SQL_ClearTable()`, `SQL_DropTable()`, `SQL_Query_Push()`, `SQL_Query()`, `SQL_SaveTable()`, `SQL_Server_DBConnection()`, `SQL_UpdateTable()`, `TimeSeriesMelt()`

Examples

```
## Not run:
data <- RemixAutoML::FakeDataGenerator(
   Correlation = 0.70,
   N = 1000L,
   ID = 2L,
   FactorCount = 2L,
   AddDate = TRUE,
   ZIP = 2L,
   TimeSeries = FALSE,
   ChainLadderData = FALSE,
   Classification = FALSE,
   MultiClass = FALSE)

## End(Not run)
```

FinalBuildArfima *FinalBuildArfima*

Description

FinalBuildArfima to generate forecasts and ensemble data

Usage

```
FinalBuildArfima(
  ModelOutputGrid = NULL,
  TimeSeriesPrepareOutput = NULL,
  FCPeriods = 1,
  MetricSelection = "MAE",
  NumberModelsScore = 1,
  ByDataType = TRUE
)
```

Arguments

`ModelOutputGrid`
: Pass along the grid output from ParallelOptimzeArima()

`TimeSeriesPrepareOutput`
: Output from TimeSeriesPrepare()

`FCPeriods`
: The number of periods ahead to forecast

`MetricSelection`
: The value returned from TimeSeriesPrepare()

`NumberModelsScore`
: The value returned from TimeSeriesPrepare()

`ByDataType`
: Set to TRUE if you want to have models represented from all data sets utilized in training

Value

Time series data sets to pass onto auto modeling functions

Author(s)

Adrian Antico

See Also

Other Time Series Helper: `FinalBuildArima()`, `FinalBuildETS()`, `FinalBuildNNET()`, `FinalBuildTBATS()`, `FinalBuildTSLM()`, `GenerateParameterGrids()`, `OptimizeArfima()`, `OptimizeArima()`, `OptimizeETS()`, `OptimizeNNET()`, `OptimizeTBATS()`, `OptimizeTSLM()`, `ParallelAutoARIMA()`, `ParallelAutoArfima()`, `ParallelAutoETS()`, `ParallelAutoNNET()`, `ParallelAutoTBATS()`, `ParallelAutoTSLM()`, `PredictArima()`, `RL_Performance()`, `Regular_Performance()`, `StackedTimeSeriesEnsembleForecast()`, `TimeSeriesDataPrepare`
`WideTimeSeriesEnsembleForecast()`

Examples

```
## Not run:
FinalBuildArfima(
  Output = NULL,
  TimeSeriesPrepareOutput = NULL,
  MaxFourierTerms = 0,
  TrainValidateShare = c(0.50,0.50),
  MaxNumberModels = 5,
  MaxRunMinutes = 5)

## End(Not run)
```

FinalBuildArima *FinalBuildArima*

Description

FinalBuildArima to generate forecasts and ensemble data

Usage

```
FinalBuildArima(
  ModelOutputGrid = NULL,
  TimeSeriesPrepareOutput = NULL,
  FCPeriods = 1,
  MetricSelection = "MAE",
  NumberModelsScore = 1,
  ByDataType = TRUE,
  DebugMode = FALSE
)
```

Arguments

ModelOutputGrid	Pass along the grid output from ParallelOptimzeArima()
TimeSeriesPrepareOutput	Output from TimeSeriesPrepare()
FCPeriods	The number of periods ahead to forecast
MetricSelection	The value returned from TimeSeriesPrepare()
NumberModelsScore	The value returned from TimeSeriesPrepare()
ByDataType	Set to TRUE if you want to have models represented from all data sets utilized in training
DebugMode	Debugging

Value

Time series data sets to pass onto auto modeling functions

Author(s)

Adrian Antico

See Also

Other Time Series Helper: FinalBuildArfima(), FinalBuildETS(), FinalBuildNNET(), FinalBuildTBATS(), FinalBuildTSLM(), GenerateParameterGrids(), OptimizeArfima(), OptimizeArima(), OptimizeETS(), OptimizeNNET(), OptimizeTBATS(), OptimizeTSLM(), ParallelAutoARIMA(), ParallelAutoArfima(), ParallelAutoETS(), ParallelAutoNNET(), ParallelAutoTBATS(), ParallelAutoTSLM(), PredictArima(), RL_Performance(), Regular_Performance(), StackedTimeSeriesEnsembleForecast(), TimeSeriesDataPrepare WideTimeSeriesEnsembleForecast()

Examples

```
## Not run:
FinalBuildArima(
  Output = NULL,
  TimeSeriesPrepareOutput = NULL,
  MaxFourierTerms = 0,
  TrainValidateShare = c(0.50,0.50),
  MaxNumberModels = 5,
  MaxRunMinutes = 5)

## End(Not run)
```

FinalBuildETS *FinalBuildETS*

Description

FinalBuildETS to generate forecasts and ensemble data

Usage

```
FinalBuildETS(
  ModelOutputGrid = NULL,
  TimeSeriesPrepareOutput = NULL,
  FCPeriods = 1,
  MetricSelection = "MAE",
  NumberModelsScore = 12,
  ByDataType = TRUE
)
```

Arguments

ModelOutputGrid	Pass along the grid output from ParallelOptimzeArima()
TimeSeriesPrepareOutput	Output from TimeSeriesPrepare()
FCPeriods	The number of periods ahead to forecast
MetricSelection	The value returned from TimeSeriesPrepare()

NumberModelsScore
: The value returned from TimeSeriesPrepare()

ByDataType
: Set to TRUE if you want to have models represented from all data sets utilized in training

Value

Time series data sets to pass onto auto modeling functions

Author(s)

Adrian Antico

See Also

Other Time Series Helper: FinalBuildArfima(), FinalBuildArima(), FinalBuildNNET(), FinalBuildTBATS(), FinalBuildTSLM(), GenerateParameterGrids(), OptimizeArfima(), OptimizeArima(), OptimizeETS(), OptimizeNNET(), OptimizeTBATS(), OptimizeTSLM(), ParallelAutoARIMA(), ParallelAutoArfima(), ParallelAutoETS(), ParallelAutoNNET(), ParallelAutoTBATS(), ParallelAutoTSLM(), PredictArima(), RL_Performance(), Regular_Performance(), StackedTimeSeriesEnsembleForecast(), TimeSeriesDataPrepare WideTimeSeriesEnsembleForecast()

Examples

```
## Not run:
FinalBuildETS(
  Output = NULL,
  TimeSeriesPrepareOutput = NULL,
  MaxFourierTerms = 0,
  TrainValidateShare = c(0.50,0.50),
  MaxNumberModels = 5,
  MaxRunMinutes = 5)

## End(Not run)
```

FinalBuildNNET *FinalBuildNNET*

Description

FinalBuildNNET to generate forecasts and ensemble data

Usage

```
FinalBuildNNET(
  ModelOutputGrid = NULL,
  TimeSeriesPrepareOutput = NULL,
  FCPeriods = 1,
  MetricSelection = "MAE",
  NumberModelsScore = 1,
  ByDataType = TRUE
)
```

Arguments

ModelOutputGrid
: Pass along the grid output from ParallelOptimzeArima()

TimeSeriesPrepareOutput
: Output from TimeSeriesPrepare()

FCPeriods
: The number of periods ahead to forecast

MetricSelection
: The value returned from TimeSeriesPrepare()

NumberModelsScore
: The value returned from TimeSeriesPrepare()

ByDataType
: Set to TRUE if you want to have models represented from all data sets utilized in training

Value

Time series data sets to pass onto auto modeling functions

Author(s)

Adrian Antico

See Also

Other Time Series Helper: FinalBuildArfima(), FinalBuildArima(), FinalBuildETS(), FinalBuildTBATS(), FinalBuildTSLM(), GenerateParameterGrids(), OptimizeArfima(), OptimizeArima(), OptimizeETS(), OptimizeNNET(), OptimizeTBATS(), OptimizeTSLM(), ParallelAutoARIMA(), ParallelAutoArfima(), ParallelAutoETS(), ParallelAutoNNET(), ParallelAutoTBATS(), ParallelAutoTSLM(), PredictArima(), RL_Performance(), Regular_Performance(), StackedTimeSeriesEnsembleForecast(), TimeSeriesDataPrepare
WideTimeSeriesEnsembleForecast()

Examples

```
## Not run:
FinalBuildNNET(
  Output = NULL,
  TimeSeriesPrepareOutput = NULL,
  MaxFourierTerms = 0,
  TrainValidateShare = c(0.50,0.50),
  MaxNumberModels = 5,
  MaxRunMinutes = 5)

## End(Not run)
```

FinalBuildTBATS *FinalBuildTBATS*

Description

FinalBuildTBATS to generate forecasts and ensemble data

Usage

```
FinalBuildTBATS(
  ModelOutputGrid = NULL,
  TimeSeriesPrepareOutput = NULL,
  FCPeriods = 1,
  MetricSelection = "MAE",
  NumberModelsScore = 1,
  ByDataType = TRUE
)
```

Arguments

ModelOutputGrid	Pass along the grid output from ParallelOptimzeArima()
TimeSeriesPrepareOutput	Output from TimeSeriesPrepare()
FCPeriods	The number of periods ahead to forecast
MetricSelection	The value returned from TimeSeriesPrepare()
NumberModelsScore	The value returned from TimeSeriesPrepare()
ByDataType	Set to TRUE if you want to have models represented from all data sets utilized in training

Value

Time series data sets to pass onto auto modeling functions

Author(s)

Adrian Antico

See Also

Other Time Series Helper: FinalBuildArfima(), FinalBuildArima(), FinalBuildETS(), FinalBuildNNET(), FinalBuildTSLM(), GenerateParameterGrids(), OptimizeArfima(), OptimizeArima(), OptimizeETS(), OptimizeNNET(), OptimizeTBATS(), OptimizeTSLM(), ParallelAutoARIMA(), ParallelAutoArfima(), ParallelAutoETS(), ParallelAutoNNET(), ParallelAutoTBATS(), ParallelAutoTSLM(), PredictArima(), RL_Performance(), Regular_Performance(), StackedTimeSeriesEnsembleForecast(), TimeSeriesDataPrepare
WideTimeSeriesEnsembleForecast()

Examples

```
## Not run:
FinalBuildTBATS(
  Output = NULL,
  TimeSeriesPrepareOutput = NULL,
  MaxFourierTerms = 0,
  TrainValidateShare = c(0.50,0.50),
  MaxNumberModels = 5,
  MaxRunMinutes = 5)

## End(Not run)
```

`FinalBuildTSLM` *FinalBuildTSLM*

Description

FinalBuildTSLM to generate forecasts and ensemble data

Usage

```
FinalBuildTSLM(
  ModelOutputGrid = NULL,
  TimeSeriesPrepareOutput = NULL,
  FCPeriods = 1,
  MetricSelection = "MAE",
  NumberModelsScore = 1,
  ByDataType = TRUE
)
```

Arguments

`ModelOutputGrid`	Pass along the grid output from ParallelOptimzeArima()
`TimeSeriesPrepareOutput`	Output from TimeSeriesPrepare()
`FCPeriods`	The number of periods ahead to forecast
`MetricSelection`	The value returned from TimeSeriesPrepare()
`NumberModelsScore`	The value returned from TimeSeriesPrepare()
`ByDataType`	Set to TRUE if you want to have models represented from all data sets utilized in training

Value

Time series data sets to pass onto auto modeling functions

Author(s)

Adrian Antico

See Also

Other Time Series Helper: `FinalBuildArfima()`, `FinalBuildArima()`, `FinalBuildETS()`, `FinalBuildNNET()`, `FinalBuildTBATS()`, `GenerateParameterGrids()`, `OptimizeArfima()`, `OptimizeArima()`, `OptimizeETS()`, `OptimizeNNET()`, `OptimizeTBATS()`, `OptimizeTSLM()`, `ParallelAutoARIMA()`, `ParallelAutoArfima()`, `ParallelAutoETS()`, `ParallelAutoNNET()`, `ParallelAutoTBATS()`, `ParallelAutoTSLM()`, `PredictArima()`, `RL_Performance()`, `Regular_Performance()`, `StackedTimeSeriesEnsembleForecast()`, `TimeSeriesDataPrepare`
`WideTimeSeriesEnsembleForecast()`

Examples

```
## Not run:
FinalBuildTSLM(
  Output = NULL,
  TimeSeriesPrepareOutput = NULL,
  MaxFourierTerms = 0,
  TrainValidateShare = c(0.50,0.50),
  MaxNumberModels = 5,
  MaxRunMinutes = 5)

## End(Not run)
```

FullFactorialCatFeatures *FullFactorialCatFeatures*

Description

FullFactorialCatFeatures reverses the difference

Usage

```
FullFactorialCatFeatures(GroupVars = GroupVariables, BottomsUp = TRUE)
```

Arguments

GroupVars	Character vector of categorical columns to fully interact
BottomsUp	TRUE or FALSE. TRUE starts with the most comlex interaction to the main effects

Author(s)

Adrian Antico

See Also

Other Data Wrangling: AutoDataDictionaries(), ColumnSubsetDataTable(), DataDisplayMeta(), FakeDataGenerator(), SQL_ClearTable(), SQL_DropTable(), SQL_Query_Push(), SQL_Query(), SQL_SaveTable(), SQL_Server_DBConnection(), SQL_UpdateTable(), TimeSeriesMelt()

`GenerateParameterGrids` *GenerateParameterGrids creates and stores model results in Experiment Grid*

Description

GenerateParameterGrids creates and stores model results in Experiment Grid

Usage

```
GenerateParameterGrids(
  Model = NULL,
  test = NULL,
  MinVal = NULL,
  DataSetName = NULL,
  SeasonalDifferences = NULL,
  SeasonalMovingAverages = NULL,
  SeasonalLags = NULL,
  MaxFourierTerms = NULL,
  Differences = NULL,
  MovingAverages = NULL,
  Lags = NULL
)
```

Arguments

`Model`	'arima', 'ets', 'tbats', 'nnet', 'arfima', 'tslm', 'dshw'
`test`	validation data
`MinVal`	Minimum value of time series
`DataSetName`	Passthrough
`SeasonalDifferences`	Passthrough
`SeasonalMovingAverages`	Passthrough
`SeasonalLags`	Passthrough
`MaxFourierTerms`	Passthrough
`Differences`	Passthrough
`MovingAverages`	Passthrough
`Lags`	Passthrough

Author(s)

Adrian Antico

See Also

Other Time Series Helper: FinalBuildArfima(), FinalBuildArima(), FinalBuildETS(), FinalBuildNNET(), FinalBuildTBATS(), FinalBuildTSLM(), OptimizeArfima(), OptimizeArima(), OptimizeETS(), OptimizeNNET(), OptimizeTBATS(), OptimizeTSLM(), ParallelAutoARIMA(), ParallelAutoArfima(), ParallelAutoETS(), ParallelAutoNNET(), ParallelAutoTBATS(), ParallelAutoTSLM(), PredictArima(), RL_Performance(), Regular_Performance(), StackedTimeSeriesEnsembleForecast(), TimeSeriesDataPrepare WideTimeSeriesEnsembleForecast()

`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",
  GroupVars = NULL,
  DateVar = "DATE",
  HighThreshold = 1.96,
  LowThreshold = -1.96,
  KeepAllCols = TRUE,
  IsDataScaled = FALSE
)
```

Arguments

`data`	the source residuals data.table
`ValueCol`	the numeric column to run anomaly detection over
`GroupVars`	this is a 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(), H2oIsolationForest(), ResidualOutliers()

Examples

```
## Not run:
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,]]
data[, Fact1 := sample(letters, size = 10000, replace = TRUE)]
data[, Fact2 := sample(letters, size = 10000, replace = TRUE)]
data[, Fact3 := sample(letters, size = 10000, replace = TRUE)]
stuff <- GenTSAnomVars(
  data,
  ValueCol = "Target",
  GroupVars = c("Fact1","Fact2","Fact3"),
  DateVar = "DateTime",
  HighThreshold = 1.96,
  LowThreshold = -1.96,
  KeepAllCols = TRUE,
  IsDataScaled  = FALSE)

## End(Not run)
```

H2oAutoencoder   *H2oAutoencoder for anomaly detection and dimensionality reduction*

Description

H2oAutoencoder for anomaly detection and or dimensionality reduction

Usage

```
H2oAutoencoder(
  AnomalyDetection = TRUE,
  DimensionReduction = TRUE,
  data,
  ValidationData = NULL,
  Features = NULL,
  RemoveFeatures = FALSE,
  NThreads = max(1L, parallel::detectCores() - 2L),
```

```
  MaxMem = "28G",
  H2oShutdown = TRUE,
  ModelID = "TestModel",
  LayerStructure = NULL,
  ReturnLayer = 4L,
  per_feature = TRUE,
  Activation = "Tanh",
  Epochs = 5L,
  L2 = 0.1,
  ElasticAveraging = TRUE,
  ElasticAveragingMovingRate = 0.9,
  ElasticAveragingRegularization = 0.001
)
```

Arguments

`AnomalyDetection`
: Set to TRUE to run anomaly detection

`DimensionReduction`
: Set to TRUE to run dimension reduction

`data`
: The data.table with the columns you wish to have analyzed

`ValidationData`
: The data.table with the columns you wish to have scored

`Features`
: NULL Column numbers or column names

`RemoveFeatures`
: Set to TRUE if you want the features you specify in the Features argument to be removed from the data returned

`NThreads`
: max(1L, parallel::detectCores()-2L)

`MaxMem`
: "28G"

`H2oShutdown`
: Setting to TRUE will shutdown H2O when it done being used internally.

`ModelID`
: "TestModel"

`LayerStructure`
: a

`ReturnLayer`
: Which layer of the NNet to return. Choose from 1-7 with 4 being the layer with the least amount of nodes

`per_feature`
: Set to TRUE to have per feature anomaly detection generated. Otherwise and overall value will be generated

`Activation`
: Choose from "Tanh", "TanhWithDropout", "Rectifier", "RectifierWithDropout","Maxout", "MaxoutWithDropout"

`Epochs`
: Quantile value to find the cutoff value for classifying outliers

`L2`
: Specify the amount of memory to allocate to H2O. E.g. "28G"

`ElasticAveraging`
: Specify the number of threads (E.g. cores * 2)

`ElasticAveragingMovingRate`
: Specify the number of decision trees to build

`ElasticAveragingRegularization`
: Specify the row sample rate per tree

Value

A data.table

Author(s)

Adrian Antico

See Also

Other Feature Engineering: AutoDataPartition(), AutoHierarchicalFourier(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), ContinuousTimeDataGenerator(), CreateCalendarVariables(), CreateHolidayVariables(), DT_GDL_Feature_Engineering(), DummifyDT(), ModelDataPrep(), Partial_DT_GDL_Feature_Engineering(), TimeSeriesFill()

Examples

```
## Not run:

# Create simulated data

# Define correlation strength of features to target
Correl <- 0.85

# Number of rows you want returned
N <- 10000

# Create data
data <- data.table::data.table(Adrian = runif(N))
data[, x1 := qnorm(Adrian)]
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(
 data.table::fifelse(Independent_Variable2 < 0.15, "A",
        data.table::fifelse(Independent_Variable2 < 0.45, "B",
               data.table::fifelse(Independent_Variable2 < 0.65,  "C",
                      data.table::fifelse(Independent_Variable2 < 0.85,  "D", "E")))))]
data.table::set(data, j = c("x1", "x2"), value = NULL)

# Get number of columns for LayerStructure
N <- length(names(data)[2L:ncol(data)])

# Run algo
Output <- RemixAutoML::H2oAutoencoder(

   # Select the service
   AnomalyDetection = TRUE,
   DimensionReduction = TRUE,

   # Data related args
   data = data,
```

```
    ValidationData = NULL,
    Features = names(data)[2L:ncol(data)],
    RemoveFeatures = FALSE,

    # H2O args
    NThreads = max(1L, parallel::detectCores()-2L),
    MaxMem = "28G",
    H2oShutdown = TRUE,
    ModelID = "TestModel",
    LayerStructure = NULL,
    ReturnLayer = 4L,
    per_feature = TRUE,
    Activation = "Tanh",
    Epochs = 5L,
    L2 = 0.10,
    ElasticAveraging = TRUE,
    ElasticAveragingMovingRate = 0.90,
    ElasticAveragingRegularization = 0.001)

  # Inspect output
  Data <- Output$Data
  Model <- Output$Model

  # If ValidationData is not null
  ValidationData <- Output$ValidationData

## End(Not run)
```

`H2oIsolationForest` *H2oIsolationForest for anomaly detection*

Description

H2oIsolationForest for anomaly detection

Usage

```
H2oIsolationForest(
  data,
  TestData = NULL,
  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
`TestData`	Data for scoring the trained isolation forest
`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 Unsupervised Learning: AutoKMeans(), GenTSAnomVars(), ResidualOutliers()

Examples

```
## Not run:

# Create simulated data

# Define correlation strength of features to target
Correl <- 0.85

# Number of rows you want returned
N <- 10000L

# Create data
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(
 data.table::fifelse(Independent_Variable2 < 0.20, "A",
        data.table::fifelse(Independent_Variable2 < 0.40, "B",
               data.table::fifelse(Independent_Variable2 < 0.6,  "C",
                      data.table::fifelse(Independent_Variable2 < 0.8,  "D", "E")))))]
data[, Independent_Variable11 := as.factor(
 data.table::fifelse(Independent_Variable2 < 0.15, "A",
        data.table::fifelse(Independent_Variable2 < 0.45, "B",
               data.table::fifelse(Independent_Variable2 < 0.65,  "C",
                      data.table::fifelse(Independent_Variable2 < 0.85,  "D", "E")))))]
data.table::set(data, j = c("x1", "x2"), value = NULL)
```

```
# Run algo
Outliers <- H2oIsolationForest(data,
                               TestData = NULL,
                               ColumnNumbers = NULL,
                               Threshold = 0.95,
                               MaxMem = "28G",
                               NThreads = -1,
                               NTrees = 100,
                               SampleRate = (sqrt(5)-1)/2)

## End(Not run)
```

ID_BuildTrainDataSets *ID_BuildTrainDataSets for assembling data*

Description

ID_BuildTrainDataSets for assembling data for the IntermittentDemandBootStrapper() function.

Usage

```
ID_BuildTrainDataSets(
  MetaData,
  data,
  Case = 2L,
  TargetVariableName = NULL,
  DateVariableName = NULL,
  GroupingVariables = NULL,
  FC_Periods,
  TimeUnit = "week",
  PowerRate = 0.5,
  SampleRate = 5L,
  TargetWindowSamples = 5L
)
```

Arguments

MetaData	This is the metadata returned from the ID_MetadataGenerator() function
data	This is your transactional data
Case	Indicate which data constructor method to use
TargetVariableName	Your target variable names
DateVariableName	Your date variable names
GroupingVariables	Your grouping variables
FC_Periods	The number of periods to forecast
TimeUnit	The time period unit, such as "day", "week", or "month"

`PowerRate`	The calculated for determining the total samples is number of records to the power of PowerRate. Then that values is multiplied by the SampleRate. This ensures that a more representative sample is generated across the data set.
`SampleRate`	The value used to sample from each level of the grouping variables
`TargetWindowSamples`	The number of different targets to utilize for a single random start date

Value

Returns the count modeling data and the size modeling data

See Also

Other Feature Engineering Helper: `AutoFourierFeatures()`, `ID_MetadataGenerator()`, `ID_TrainingDataGenerato` `ID_TrainingDataGenerator()`

`ID_MetadataGenerator` *ID_MetadataGenerator for summary metadata for transactional data*

Description

ID_MetadataGenerator for summary metadata for transactional data. The data returned from this function feeds into the IntermittentDemandBootStrapper() function.

Usage

```
ID_MetadataGenerator(
  data,
  RestrictDateRange = TRUE,
  DateVariableName = NULL,
  GroupingVariables = NULL,
  MinTimeWindow = 1L,
  MinTxnRecords = 2L,
  DateInterval = "day"
)
```

Arguments

`data`	This is your transactional level data
`RestrictDateRange`	= TRUE
`DateVariableName`	Bla
`GroupingVariables`	Bla
`MinTimeWindow`	The number of time periods you would like to omit for training. Default is 1 so that at a minimum, there is at least one period of values to forecast. You can set it up to a larger value if you do not want more possible target windows for the lower target window values.
`MinTxnRecords`	I typically set this to 2 so that there is at least one other instance of demand so that the forecasted values are not complete nonsense.
`DateInterval`	This is the time unit for determining date calculations

Value

Returns a data.table with summary information for the IntermittentDemandBootStrapper() function.

See Also

Other Feature Engineering Helper: AutoFourierFeatures(), ID_BuildTrainDataSets(), ID_TrainingDataGenerat
ID_TrainingDataGenerator()

Examples

```
## Not run:
# Generate Metadata----
MetaData <- ID_MetadataGenerator(
  data = data,
  RestrictDateRange = TRUE,
  DateVariableName = DateVariableName,
  GroupingVariables = GroupingVariables,
  MinTimeWindow = MinTimeWindow,
  MinTxnRecords = MinTxnRecords,
  DateInterval = TimeUnit,
  TimeUnit = TimeUnit
)

## End(Not run)
```

ID_TrainingDataGenerator

ID_TrainingDataGenerator for subsetting data

Description

ID_TrainingDataGenerator for subsetting data for the IntermittentDemandBootStrapper() function.

Usage

```
ID_TrainingDataGenerator(
  data,
  Type = "timetoevent1",
  TargetVariableName = NULL,
  Level = NULL,
  DateVariableName = NULL,
  GroupingVariables = NULL,
  RandomStartDate = NULL,
  TimeUnit = NULL,
  TargetWindow = NULL
)
```

Arguments

data	Source data
Type	"timetoevent1", "eventinwindow1"

`TargetVariableName`
: Name of the variables to run feature engineering on. List the actual target variable name first.

`Level`
: The individual level of your group variable

`DateVariableName`
: Name of your date variable

`GroupingVariables`
: Your grouping variables

`RandomStartDate`
: The date to partition the data

`TimeUnit`
: This is the TimeUnit you selected for aggregation

`TargetWindow`
: The length of the target window sampled

Value

Returns two data sets for the IntermittentDemandBootStrapper() function based on a single level from the grouping variables.

See Also

Other Feature Engineering Helper: AutoFourierFeatures(), ID_BuildTrainDataSets(), ID_MetadataGenerator(), ID_TrainingDataGenerator2()

ID_TrainingDataGenerator2
ID_TrainingDataGenerator2 for subsetting data

Description

ID_TrainingDataGenerator2 for subsetting data for the IntermittentDemandBootStrapper() function.

Usage

```
ID_TrainingDataGenerator2(
  data,
  TargetVariableName = NULL,
  Level = NULL,
  GroupingVariables = NULL,
  DateVariableName = NULL,
  RandomStartDate = NULL,
  TimeUnit = NULL,
  TargetWindow = NULL
)
```

Arguments

`data`	Source data
`TargetVariableName`	vector of variable names
`Level`	The individual level of your group variable
`GroupingVariables`	Your grouping variables
`DateVariableName`	Name of your date variable
`RandomStartDate`	The date to partition the data
`TimeUnit`	This is the TimeUnit you selected for aggregation
`TargetWindow`	The length of the target window sampled

Value

Returns two data sets for the IntermittentDemandBootStrapper() function based on a single level from the grouping variables.

See Also

Other Feature Engineering Helper: `AutoFourierFeatures()`, `ID_BuildTrainDataSets()`, `ID_MetadataGenerator()`, `ID_TrainingDataGenerator()`

`IntermittentDemandScoringDataGenerator`
IntermittentDemandScoringDataGenerator

Description

IntermittentDemandScoringDataGenerator creates the scoring data for forecasting. It will recreate the same features used for modeling, take the most recent record, and then duplicate those records for each forecast period specifed.

Usage

```
IntermittentDemandScoringDataGenerator(
  data = NULL,
  FC_Periods = 52,
  SaveData = FALSE,
  FilePath = NULL,
  TargetVariableName = "qty",
  DateVariableName = "date",
  GroupingVariables = "sku",
  Lags = 1:7,
  MovingAverages = seq(7, 28, 7),
  TimeTrendVariable = TRUE,
  TimeUnit = "day",
  CurrentDate = NULL,
```

```
  CalendarVariables = c("wday", "mday", "yday", "week", "isoweek", "month", "quarter",
    "year"),
   HolidayGroups = "USPublicHolidays"
)
```

Arguments

data	This is your source data
FC_Periods	The number of periods you set up to forecast
SaveData	Set to TRUE to save the output data to file
FilePath	Set a path file have the data saved there
TargetVariableName	Name or column number of your target variable
DateVariableName	Name or column number of your date variable
GroupingVariables	Name or column number of your group variables
Lags	The number of lags used in building the modeling data sets
MovingAverages	The number of moving averages used in building the modeling data sets
TimeTrendVariable	Set to TRUE if you did so in model data creation
TimeUnit	Set to the same time unit used in modeling data creation
CurrentDate	Set this to the current date or a date that you want. It is user specified in case you want to score historical data.
CalendarVariables	Set this to the same setting you used in modeling data creation
HolidayGroups	Set this to the same setting you used in modeling data creation

Value

Returns the most recent records for every level of your grouping variables with all the feature used in model building

Author(s)

Adrian Antico

See Also

Other Automated Model Scoring: AutoCatBoostScoring(), AutoH2OMLScoring(), AutoH2OModeler(), AutoHurdleScoring(), AutoXGBoostScoring()

Examples

```
## Not run:
 ScoringData <- IntermittentDemandScoringDataGenerator(
   data = data,
   SaveData = FALSE,
   FilePath = NULL,
   TargetVariableName = "qty",
   DateVariableName = "date",
```

```
    GroupingVariables = "sku",
    Lags = 1:7,
    MovingAverages = seq(7,28,7),
    TimeTrendVariable = TRUE,
    TimeUnit = "day",
    CurrentDate = NULL,
    CalendarVariables = c("wday",
                          "mday",
                          "yday",
                          "week",
                          "isoweek",
                          "month",
                          "quarter",
                          "year"),
    HolidayGroups = "USPublicHolidays")

## End(Not run)
```

`LimeModel` *LimeModel to build a lime model*

Description

LimeModel to build a lime model for prediction explanations in this package#'

Usage

```
LimeModel(
  data,
  Model = NULL,
  Bins = 10,
  ModelType = "xgboost",
  NThreads = parallel::detectCores(),
  MaxMem = "32G",
  ModelPath = NULL,
  ModelID = NULL
)
```

Arguments

`data`	Supply a training data set. This data set should be the data right before it gets converted to an h2o, catboost, or xgboost data object.
`Model`	Supply the model returned from training with the Auto__() functions.
`Bins`	Number of bins for discretizing numeric features
`ModelType`	Select from xgboost, h2o, and catboost
`NThreads`	Number of CPU threads
`MaxMem`	For use with H2O models. E.g. set to "28G"
`ModelPath`	Set to the path where your ML model is saved
`ModelID`	ID used to identify your ML model

Value

Model for utilizing lime

Author(s)

Adrian Antico

See Also

Other Model Evaluation and Interpretation: AutoLimeAid(), EvalPlot(), ParDepCalPlots(), RedYellowGreen(), threshOptim()

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,
  FactorToChar = FALSE,
  IntToNumeric = TRUE,
  DateToChar = FALSE,
  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
FactorToChar	Converts to character
IntToNumeric	Defaults to TRUE which tells the function to convert integers to numeric
DateToChar	Converts date columns into character columns
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(), AutoHierarchicalFourier(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), ContinuousTimeDataGenerator(), CreateCalendarVariables(), CreateHolidayVariables(), DT_GDL_Feature_Engineering(), DummifyDT(), H2oAutoencoder(), Partial_DT_GDL_Feature_Engineering(), TimeSeriesFill()

Examples

```
## Not run:
# Create fake data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.75,
  N = 250000L,
  ID = 2L,
  ZIP = 0L,
  FactorCount = 6L,
  AddDate = TRUE,
  Classification = FALSE,
  MultiClass = FALSE)

# Check column types
str(data)

# Convert some factors to character
data <- RemixAutoML::ModelDataPrep(
  data,
  Impute       = TRUE,
  CharToFactor = FALSE,
  FactorToChar = TRUE,
  IntToNumeric = TRUE,
  DateToChar   = FALSE,
  RemoveDates  = TRUE,
  MissFactor   = "0",
  MissNum      = -1,
  IgnoreCols   = c("Factor_1"))

# Check column types
str(data)

## End(Not run)
```

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 Graphics: `RemixTheme()`, `TimeSeriesPlotter()`

Examples

```
## Not run:
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)

## End(Not run)
```

OptimizeArfima *OptimizeArfima is a function that takes raw data and returns time series data*

Description

OptimizeArfima is a function that takes raw data and returns the necessary time series data and objects for model building. It also fills any time gaps with zeros. Use this before you run any time series model functions.

Usage

```
OptimizeArfima(
  Output,
  MetricSelection = "MAE",
  DataSetName = NULL,
  train = NULL,
  test = NULL,
  Lags = NULL,
  MovingAverages = NULL,
  FullData = NULL,
  HoldOutPeriods = NULL,
  MinVal = NULL,
  TargetName = NULL,
  DateName = NULL,
  TrainValidateShare = NULL,
  FinalGrid = NULL
)
```

Arguments

`Output`	This is passed through as output from TimeSeriesDataPrepare() and passed through ParallelArima()
`MetricSelection`	Select from "MSE", "MAE", or "MAPE"
`DataSetName`	This is the name of the data set passed through in parallel loop
`train`	Training data returned from TimeSeriesDataPrepare()
`test`	Test data returned from TimeSeriesDataPrepare()
`Lags`	Max lags
`MovingAverages`	Max moving averages

FullData Full series data for scoring and ensemble

HoldOutPeriods Holdout periods returned from TimeSeriesDataPrepare()

MinVal Minimum value of target variable returned from TimeSeriesDataPrepare()

TargetName Target variable name returned from TimeSeriesDataPrepare()

DateName Date variable name returned from TimeSeriesDataPrepare()

TrainValidateShare
A two-element numeric vector. The first element is the weight applied to the training performance and the remainder is applied to the validation performance.

FinalGrid Grid for forecasting models

Value

Time series data sets to pass onto auto modeling functions

Author(s)

Adrian Antico

See Also

Other Time Series Helper: FinalBuildArfima(), FinalBuildArima(), FinalBuildETS(), FinalBuildNNET(), FinalBuildTBATS(), FinalBuildTSLM(), GenerateParameterGrids(), OptimizeArima(), OptimizeETS(), OptimizeNNET(), OptimizeTBATS(), OptimizeTSLM(), ParallelAutoARIMA(), ParallelAutoArfima(), ParallelAutoETS(), ParallelAutoNNET(), ParallelAutoTBATS(), ParallelAutoTSLM(), PredictArima(), RL_Performance(), Regular_Performance(), StackedTimeSeriesEnsembleForecast(), TimeSeriesDataPrepare
WideTimeSeriesEnsembleForecast()

Examples

```
## Not run:
Results <- OptimizeArfima(
  Output,
  MetricSelection = "MAE",
  DataSetName = NULL,
  train = NULL,
  test = NULL,
  Lags = NULL,
  MovingAverages = NULL,
  FullData = NULL,
  HoldOutPeriods = NULL,
  MinVal = NULL,
  TargetName = NULL,
  DateName = NULL,
  TrainValidateShare = NULL,
  FinalGrid = NULL)

## End(Not run)
```

OptimizeArima — *OptimizeArima is a function that takes raw data and returns time series data*

Description

OptimizeArima is a function that takes raw data and returns the necessary time series data and objects for model building. It also fills any time gaps with zeros. Use this before you run any time series model functions.

Usage

```
OptimizeArima(
  Output,
  MetricSelection = "MAE",
  DataSetName = NULL,
  train = NULL,
  test = NULL,
  FullData = NULL,
  HoldOutPeriods = NULL,
  MinVal = NULL,
  TargetName = NULL,
  DateName = NULL,
  Lags = NULL,
  SeasonalLags = NULL,
  MovingAverages = NULL,
  SeasonalMovingAverages = NULL,
  Differences = NULL,
  SeasonalDifferences = NULL,
  MaxFourierTerms = NULL,
  TrainValidateShare = NULL,
  MaxRunsWithoutNewWinner = 20,
  MaxNumberModels = NULL,
  MaxRunMinutes = NULL,
  FinalGrid = NULL,
  DebugMode = FALSE
)
```

Arguments

`Output`	This is passed through as output from TimeSeriesDataPrepare() and passed through ParallelArima()
`MetricSelection`	Select from "MSE", "MAE", or "MAPE"
`DataSetName`	This is the name of the data set passed through in parallel loop
`train`	Training data returned from TimeSeriesDataPrepare()
`test`	Test data returned from TimeSeriesDataPrepare()
`FullData`	Full series data for scoring and ensemble
`HoldOutPeriods`	Holdout periods returned from TimeSeriesDataPrepare()

`MinVal`	Minimum value of target variable returned from TimeSeriesDataPrepare()
`TargetName`	Target variable name returned from TimeSeriesDataPrepare()
`DateName`	Date variable name returned from TimeSeriesDataPrepare()
`Lags`	Max value of lag returned from TimeSeriesDataPrepare()
`SeasonalLags`	Max value of seasonal lags returned from TimeSeriesDataPrepare()
`MovingAverages`	Max value of moving averages
`SeasonalMovingAverages`	Max value of seasonal moving average
`Differences`	Max value of difference returned from TimeSeriesDataPrepare()
`SeasonalDifferences`	Max value of seasonal difference returned from TimeSeriesDataPrepare()
`MaxFourierTerms`	Max value of fourier pairs
`TrainValidateShare`	A two-element numeric vector. The first element is the weight applied to the training performance and the remainder is applied to the validation performance.
`MaxRunsWithoutNewWinner`	The number of runs without a new winner which if passed tells the function to stop
`MaxNumberModels`	The number of models you want to test.
`MaxRunMinutes`	Time
`FinalGrid`	If NULL, regular train optimization occurs. If the grid is supplied, final builds are conducted.
`DebugMode`	Debugging

Value

Time series data sets to pass onto auto modeling functions

Author(s)

Adrian Antico

See Also

Other Time Series Helper: `FinalBuildArfima()`, `FinalBuildArima()`, `FinalBuildETS()`, `FinalBuildNNET()`, `FinalBuildTBATS()`, `FinalBuildTSLM()`, `GenerateParameterGrids()`, `OptimizeArfima()`, `OptimizeETS()`, `OptimizeNNET()`, `OptimizeTBATS()`, `OptimizeTSLM()`, `ParallelAutoARIMA()`, `ParallelAutoArfima()`, `ParallelAutoETS()`, `ParallelAutoNNET()`, `ParallelAutoTBATS()`, `ParallelAutoTSLM()`, `PredictArima()`, `RL_Performance()`, `Regular_Performance()`, `StackedTimeSeriesEnsembleForecast()`, `TimeSeriesDataPrepare`
`WideTimeSeriesEnsembleForecast()`

Examples

```
## Not run:
Results <- OptimizeArima(
  Output,
  MetricSelection = "MAE",
  DataSetName = NULL,
```

```
  train = NULL,
  test = NULL,
  FullData = NULL,
  HoldOutPeriods = NULL,
  MinVal = NULL,
  TargetName = NULL,
  DateName = NULL,
  Lags = NULL,
  SeasonalLags = NULL,
  MovingAverages = NULL,
  SeasonalMovingAverages = NULL,
  Differences = NULL,
  SeasonalDifferences = NULL,
  MaxFourierTerms = NULL,
  TrainValidateShare = NULL,
  MaxRunsWithoutNewWinner = 20,
  MaxNumberModels = 5,
  MaxRunMinutes = NULL,
  FinalGrid = NULL)

## End(Not run)
```

OptimizeETS — *OptimizeETS is a function that takes raw data and returns time series data*

Description

OptimizeETS is a function that takes raw data and returns the necessary time series data and objects for model building. It also fills any time gaps with zeros. Use this before you run any time series model functions.

Usage

```
OptimizeETS(
  Output,
  MetricSelection = "MAE",
  DataSetName = NULL,
  train = NULL,
  test = NULL,
  FullData = NULL,
  HoldOutPeriods = NULL,
  MinVal = NULL,
  TargetName = NULL,
  DateName = NULL,
  TrainValidateShare = NULL,
  FinalGrid = NULL
)
```

Arguments

`Output`	This is passed through as output from TimeSeriesDataPrepare() and passed through ParallelArima()

MetricSelection
: Select from "MSE", "MAE", or "MAPE"

DataSetName
: This is the name of the data set passed through in parallel loop

train
: Training data returned from TimeSeriesDataPrepare()

test
: Test data returned from TimeSeriesDataPrepare()

FullData
: Full series data for scoring and ensemble

HoldOutPeriods
: Holdout periods returned from TimeSeriesDataPrepare()

MinVal
: Minimum value of target variable returned from TimeSeriesDataPrepare()

TargetName
: Target variable name returned from TimeSeriesDataPrepare()

DateName
: Date variable name returned from TimeSeriesDataPrepare()

TrainValidateShare
: A two-element numeric vector. The first element is the weight applied to the training performance and the remainder is applied to the validation performance.

FinalGrid
: Grid for forecasting models

Value

Time series data sets to pass onto auto modeling functions

Author(s)

Adrian Antico

See Also

Other Time Series Helper: FinalBuildArfima(), FinalBuildArima(), FinalBuildETS(), FinalBuildNNET(), FinalBuildTBATS(), FinalBuildTSLM(), GenerateParameterGrids(), OptimizeArfima(), OptimizeArima(), OptimizeNNET(), OptimizeTBATS(), OptimizeTSLM(), ParallelAutoARIMA(), ParallelAutoArfima(), ParallelAutoETS(), ParallelAutoNNET(), ParallelAutoTBATS(), ParallelAutoTSLM(), PredictArima(), RL_Performance(), Regular_Performance(), StackedTimeSeriesEnsembleForecast(), TimeSeriesDataPrepare
WideTimeSeriesEnsembleForecast()

Examples

```
## Not run:
Results <- OptimizeETS(
  Output,
  MetricSelection = "MAE",
  DataSetName = NULL,
  train = NULL,
  test = NULL,
  FullData = NULL,
  HoldOutPeriods = NULL,
  MinVal = NULL,
  TargetName = NULL,
  DateName = NULL,
  TrainValidateShare = NULL,
  FinalGrid = NULL)

## End(Not run)
```

`OptimizeNNET` *OptimizeNNET is a function that takes raw data and returns time series data*

Description

OptimizeNNET is a function that takes raw data and returns the necessary time series data and objects for model building. It also fills any time gaps with zeros. Use this before you run any time series model functions.

Usage

```
OptimizeNNET(
  Output,
  MetricSelection = "MAE",
  DataSetName = NULL,
  train = NULL,
  test = NULL,
  FullData = NULL,
  HoldOutPeriods = NULL,
  MinVal = NULL,
  TargetName = NULL,
  DateName = NULL,
  Lags = NULL,
  SeasonalLags = NULL,
  MaxFourierTerms = NULL,
  TrainValidateShare = NULL,
  MaxRunsWithoutNewWinner = 20,
  MaxNumberModels = NULL,
  MaxRunMinutes = NULL,
  FinalGrid = NULL
)
```

Arguments

`Output`	This is passed through as output from TimeSeriesDataPrepare() and passed through ParallelArima()
`MetricSelection`	Select from "MSE", "MAE", or "MAPE"
`DataSetName`	This is the name of the data set passed through in parallel loop
`train`	Training data returned from TimeSeriesDataPrepare()
`test`	Test data returned from TimeSeriesDataPrepare()
`FullData`	Full series data for scoring and ensemble
`HoldOutPeriods`	Holdout periods returned from TimeSeriesDataPrepare()
`MinVal`	Minimum value of target variable returned from TimeSeriesDataPrepare()
`TargetName`	Target variable name returned from TimeSeriesDataPrepare()
`DateName`	Date variable name returned from TimeSeriesDataPrepare()
`Lags`	Max value of lag returned from TimeSeriesDataPrepare()

SeasonalLags Max value of seasonal lags returned from TimeSeriesDataPrepare()

MaxFourierTerms
Max value of fourier pairs

TrainValidateShare
A two-element numeric vector. The first element is the weight applied to the training performance and the remainder is applied to the validation performance.

MaxRunsWithoutNewWinner
The number of runs without a new winner which if passed tells the function to stop

MaxNumberModels
The number of models you want to test.

MaxRunMinutes Time

FinalGrid If NULL, regular train optimization occurs. If the grid is supplied, final builds are conducted.

Value

Time series data sets to pass onto auto modeling functions

Author(s)

Adrian Antico

See Also

Other Time Series Helper: FinalBuildArfima(), FinalBuildArima(), FinalBuildETS(), FinalBuildNNET(), FinalBuildTBATS(), FinalBuildTSLM(), GenerateParameterGrids(), OptimizeArfima(), OptimizeArima(), OptimizeETS(), OptimizeTBATS(), OptimizeTSLM(), ParallelAutoARIMA(), ParallelAutoArfima(), ParallelAutoETS(), ParallelAutoNNET(), ParallelAutoTBATS(), ParallelAutoTSLM(), PredictArima(), RL_Performance(), Regular_Performance(), StackedTimeSeriesEnsembleForecast(), TimeSeriesDataPrepare WideTimeSeriesEnsembleForecast()

Examples

```
## Not run:
Results <- OptimizeNNET(
  Output,
  MetricSelection = "MAE",
  DataSetName = NULL,
  train = NULL,
  test = NULL,
  FullData = NULL,
  HoldOutPeriods = NULL,
  MinVal = NULL,
  TargetName = NULL,
  DateName = NULL,
  Lags = NULL,
  SeasonalLags = NULL,
  MaxFourierTerms = NULL,
  TrainValidateShare = NULL,
  MaxRunsWithoutNewWinner = 20,
  MaxNumberModels = 5,
  MaxRunMinutes = NULL,
  FinalGrid = NULL)
```

```
## End(Not run)
```

OptimizeTBATS — *OptimizeTBATS is a function that takes raw data and returns time series data*

Description

OptimizeTBATS is a function that takes raw data and returns the necessary time series data and objects for model building. It also fills any time gaps with zeros. Use this before you run any time series model functions.

Usage

```
OptimizeTBATS(
  Output,
  MetricSelection = "MAE",
  DataSetName = NULL,
  train = NULL,
  test = NULL,
  Lags = NULL,
  MovingAverages = NULL,
  FullData = NULL,
  HoldOutPeriods = NULL,
  MinVal = NULL,
  TargetName = NULL,
  DateName = NULL,
  TrainValidateShare = NULL,
  FinalGrid = NULL
)
```

Arguments

`Output`	This is passed through as output from TimeSeriesDataPrepare() and passed through ParallelArima()
`MetricSelection`	Select from "MSE", "MAE", or "MAPE"
`DataSetName`	This is the name of the data set passed through in parallel loop
`train`	Training data returned from TimeSeriesDataPrepare()
`test`	Test data returned from TimeSeriesDataPrepare()
`Lags`	Max lags
`MovingAverages`	Max moving averages
`FullData`	Full series data for scoring and ensemble
`HoldOutPeriods`	Holdout periods returned from TimeSeriesDataPrepare()
`MinVal`	Minimum value of target variable returned from TimeSeriesDataPrepare()
`TargetName`	Target variable name returned from TimeSeriesDataPrepare()
`DateName`	Date variable name returned from TimeSeriesDataPrepare()

TrainValidateShare
: A two-element numeric vector. The first element is the weight applied to the training performance and the remainder is applied to the validation performance.

FinalGrid
: Grid for forecasting models

Value

Time series data sets to pass onto auto modeling functions

Author(s)

Adrian Antico

See Also

Other Time Series Helper: FinalBuildArfima(), FinalBuildArima(), FinalBuildETS(), FinalBuildNNET(), FinalBuildTBATS(), FinalBuildTSLM(), GenerateParameterGrids(), OptimizeArfima(), OptimizeArima(), OptimizeETS(), OptimizeNNET(), OptimizeTSLM(), ParallelAutoARIMA(), ParallelAutoArfima(), ParallelAutoETS(), ParallelAutoNNET(), ParallelAutoTBATS(), ParallelAutoTSLM(), PredictArima(), RL_Performance(), Regular_Performance(), StackedTimeSeriesEnsembleForecast(), TimeSeriesDataPrepare WideTimeSeriesEnsembleForecast()

Examples

```
## Not run:
Results <- OptimizeTBATS(
  Output,
  MetricSelection = "MAE",
  DataSetName = NULL,
  train = NULL,
  test = NULL,
  Lags = NULL,
  MovingAverages = NULL,
  FullData = NULL,
  HoldOutPeriods = NULL,
  MinVal = NULL,
  TargetName = NULL,
  DateName = NULL,
  TrainValidateShare = NULL,
  FinalGrid = NULL)

## End(Not run)
```

OptimizeTSLM — *OptimizeTSLM is a function that takes raw data and returns time series data*

Description

OptimizeTSLM is a function that takes raw data and returns the necessary time series data and objects for model building. It also fills any time gaps with zeros. Use this before you run any time series model functions.

Usage

```
OptimizeTSLM(
  Output,
  MetricSelection = "MAE",
  DataSetName = NULL,
  train = NULL,
  test = NULL,
  FullData = NULL,
  HoldOutPeriods = NULL,
  MinVal = NULL,
  TargetName = NULL,
  DateName = NULL,
  TrainValidateShare = NULL,
  FinalGrid = NULL
)
```

Arguments

`Output`	This is passed through as output from TimeSeriesDataPrepare() and passed through ParallelArima()
`MetricSelection`	Select from "MSE", "MAE", or "MAPE"
`DataSetName`	This is the name of the data set passed through in parallel loop
`train`	Training data returned from TimeSeriesDataPrepare()
`test`	Test data returned from TimeSeriesDataPrepare()
`FullData`	Full series data for scoring and ensemble
`HoldOutPeriods`	Holdout periods returned from TimeSeriesDataPrepare()
`MinVal`	Minimum value of target variable returned from TimeSeriesDataPrepare()
`TargetName`	Target variable name returned from TimeSeriesDataPrepare()
`DateName`	Date variable name returned from TimeSeriesDataPrepare()
`TrainValidateShare`	A two-element numeric vector. The first element is the weight applied to the training performance and the remainder is applied to the validation performance.
`FinalGrid`	Grid for forecasting models

Value

Time series data sets to pass onto auto modeling functions

Author(s)

Adrian Antico

See Also

Other Time Series Helper: FinalBuildArfima(), FinalBuildArima(), FinalBuildETS(), FinalBuildNNET(), FinalBuildTBATS(), FinalBuildTSLM(), GenerateParameterGrids(), OptimizeArfima(), OptimizeArima(), OptimizeETS(), OptimizeNNET(), OptimizeTBATS(), ParallelAutoARIMA(), ParallelAutoArfima(), ParallelAutoETS(), ParallelAutoNNET(), ParallelAutoTBATS(), ParallelAutoTSLM(), PredictArima(), RL_Performance(), Regular_Performance(), StackedTimeSeriesEnsembleForecast(), TimeSeriesDataPrepare WideTimeSeriesEnsembleForecast()

Examples

```
## Not run:
Results <- OptimizeTSLM(
  Output,
  MetricSelection = "MAE",
  DataSetName = NULL,
  train = NULL,
  test = NULL,
  FullData = NULL,
  HoldOutPeriods = NULL,
  MinVal = NULL,
  TargetName = NULL,
  DateName = NULL,
  TrainValidateShare = NULL,
  FinalGrid = NULL)

## End(Not run)
```

ParallelAutoArfima *ParallelAutoArfima*

Description

ParallelAutoArfima to run the 4 data sets at once

Usage

```
ParallelAutoArfima(
  Output,
  MetricSelection = "MAE",
  TrainValidateShare = c(0.5, 0.5)
)
```

Arguments

Output	The output returned from TimeSeriesDataPrepare()
MetricSelection	Choose from MAE, MSE, and MAPE
TrainValidateShare	The value returned from TimeSeriesPrepare()

Value

Time series data sets to pass onto auto modeling functions

Author(s)

Adrian Antico

See Also

Other Time Series Helper: FinalBuildArfima(), FinalBuildArima(), FinalBuildETS(), FinalBuildNNET(), FinalBuildTBATS(), FinalBuildTSLM(), GenerateParameterGrids(), OptimizeArfima(), OptimizeArima(), OptimizeETS(), OptimizeNNET(), OptimizeTBATS(), OptimizeTSLM(), ParallelAutoARIMA(), ParallelAutoETS(), ParallelAutoNNET(), ParallelAutoTBATS(), ParallelAutoTSLM(), PredictArima(), RL_Performance(), Regular_Performance(), StackedTimeSeriesEnsembleForecast(), TimeSeriesDataPrepare WideTimeSeriesEnsembleForecast()

Examples

```
## Not run:
ParallelAutoArfima(
  MetricSelection = "MAE",
  Output = NULL,
  TrainValidateShare = c(0.50,0.50))

## End(Not run)
```

`ParallelAutoARIMA` *ParallelAutoARIMA to run the 4 data sets at once*

Description

ParallelAutoARIMA to run the 4 data sets at once

Usage

```
ParallelAutoARIMA(
  Output,
  MetricSelection = "MAE",
  MaxFourierTerms = 1L,
  TrainValidateShare = c(0.5, 0.5),
  MaxNumberModels = 20,
  MaxRunMinutes = 5L,
  MaxRunsWithoutNewWinner = 12,
  NumCores = max(1L, parallel::detectCores() - 2L)
)
```

Arguments

`Output`	The output returned from TimeSeriesDataPrepare()
`MetricSelection`	Choose from MAE, MSE, and MAPE
`MaxFourierTerms`	Fourier pairs
`TrainValidateShare`	c(0.50,0.50)
`MaxNumberModels`	20
`MaxRunMinutes`	5
`MaxRunsWithoutNewWinner`	12
`NumCores`	Value

Value

Time series data sets to pass onto auto modeling functions

Author(s)

Adrian Antico

See Also

Other Time Series Helper: FinalBuildArfima(), FinalBuildArima(), FinalBuildETS(), FinalBuildNNET(), FinalBuildTBATS(), FinalBuildTSLM(), GenerateParameterGrids(), OptimizeArfima(), OptimizeArima(), OptimizeETS(), OptimizeNNET(), OptimizeTBATS(), OptimizeTSLM(), ParallelAutoArfima(), ParallelAutoETS(), ParallelAutoNNET(), ParallelAutoTBATS(), ParallelAutoTSLM(), PredictArima(), RL_Performance(), Regular_Performance(), StackedTimeSeriesEnsembleForecast(), TimeSeriesDataPrepare WideTimeSeriesEnsembleForecast()

Examples

```
## Not run:
ParallelAutoARIMA(
  MetricSelection = "MAE",
  Output = NULL,
  MaxRunsWithoutNewWinner = 20,
  TrainValidateShare = c(0.50,0.50),
  MaxNumberModels = 5,
  MaxRunMinutes = 5)

## End(Not run)
```

ParallelAutoETS *ParallelAutoETS*

Description

ParallelAutoETS to run the 4 data sets at once

Usage

```
ParallelAutoETS(
  Output,
  MetricSelection = "MAE",
  TrainValidateShare = c(0.5, 0.5)
)
```

Arguments

Output	The output returned from TimeSeriesDataPrepare()
MetricSelection	Choose from MAE, MSE, and MAPE
TrainValidateShare	The value returned from TimeSeriesPrepare()

Value

Time series data sets to pass onto auto modeling functions

Author(s)

Adrian Antico

See Also

Other Time Series Helper: FinalBuildArfima(), FinalBuildArima(), FinalBuildETS(), FinalBuildNNET(), FinalBuildTBATS(), FinalBuildTSLM(), GenerateParameterGrids(), OptimizeArfima(), OptimizeArima(), OptimizeETS(), OptimizeNNET(), OptimizeTBATS(), OptimizeTSLM(), ParallelAutoARIMA(), ParallelAutoArfima(), ParallelAutoNNET(), ParallelAutoTBATS(), ParallelAutoTSLM(), PredictArima(), RL_Performance(), Regular_Performance(), StackedTimeSeriesEnsembleForecast(), TimeSeriesDataPrepare(), WideTimeSeriesEnsembleForecast()

Examples

```
## Not run:
ParallelAutoETS(
  MetricSelection = "MAE",
  Output = NULL,
  TrainValidateShare = c(0.50,0.50))

## End(Not run)
```

`ParallelAutoNNET` *ParallelAutoNNET to run the 4 data sets at once*

Description

ParallelAutoNNET to run the 4 data sets at once

Usage

```
ParallelAutoNNET(
  Output,
  MetricSelection = "MAE",
  MaxFourierTerms = 1,
  TrainValidateShare = c(0.5, 0.5),
  MaxNumberModels = 20,
  MaxRunMinutes = 5,
  MaxRunsWithoutNewWinner = 12
)
```

Arguments

`Output` The output returned from TimeSeriesDataPrepare()

`MetricSelection` Choose from MAE, MSE, and MAPE

`MaxFourierTerms` Fourier pairs

TrainValidateShare
c(0.50,0.50)

MaxNumberModels
20

MaxRunMinutes 5

MaxRunsWithoutNewWinner
12

Value

Time series data sets to pass onto auto modeling functions

Author(s)

Adrian Antico

See Also

Other Time Series Helper: FinalBuildArfima(), FinalBuildArima(), FinalBuildETS(), FinalBuildNNET(), FinalBuildTBATS(), FinalBuildTSLM(), GenerateParameterGrids(), OptimizeArfima(), OptimizeArima(), OptimizeETS(), OptimizeNNET(), OptimizeTBATS(), OptimizeTSLM(), ParallelAutoARIMA(), ParallelAutoArfima(), ParallelAutoETS(), ParallelAutoTBATS(), ParallelAutoTSLM(), PredictArima(), RL_Performance(), Regular_Performance(), StackedTimeSeriesEnsembleForecast(), TimeSeriesDataPrepare WideTimeSeriesEnsembleForecast()

Examples

```
## Not run:
ParallelAutoNNET(
  MetricSelection = "MAE",
  Output = NULL,
  MaxRunsWithoutNewWinner = 20,
  TrainValidateShare = c(0.50,0.50),
  MaxNumberModels = 5,
  MaxRunMinutes = 5)

## End(Not run)
```

ParallelAutoTBATS *ParallelAutoTBATS*

Description

ParallelAutoTBATS to run the 4 data sets at once

Usage

```
ParallelAutoTBATS(
  Output,
  MetricSelection = "MAE",
  TrainValidateShare = c(0.5, 0.5)
)
```

Arguments

Output — The output returned from TimeSeriesDataPrepare()

MetricSelection — Choose from MAE, MSE, and MAPE

TrainValidateShare — The value returned from TimeSeriesPrepare()

Value

Time series data sets to pass onto auto modeling functions

Author(s)

Adrian Antico

See Also

Other Time Series Helper: FinalBuildArfima(), FinalBuildArima(), FinalBuildETS(), FinalBuildNNET(), FinalBuildTBATS(), FinalBuildTSLM(), GenerateParameterGrids(), OptimizeArfima(), OptimizeArima(), OptimizeETS(), OptimizeNNET(), OptimizeTBATS(), OptimizeTSLM(), ParallelAutoARIMA(), ParallelAutoArfima(), ParallelAutoETS(), ParallelAutoNNET(), ParallelAutoTSLM(), PredictArima(), RL_Performance(), Regular_Performance(), StackedTimeSeriesEnsembleForecast(), TimeSeriesDataPrepare
WideTimeSeriesEnsembleForecast()

Examples

```
## Not run:
ParallelAutoTBATS(
  MetricSelection = "MAE",
  Output = NULL,
  TrainValidateShare = c(0.50,0.50))

## End(Not run)
```

ParallelAutoTSLM *ParallelAutoTSLM*

Description

ParallelAutoTSLM to run the 4 data sets at once

Usage

```
ParallelAutoTSLM(
  Output,
  MetricSelection = "MAE",
  TrainValidateShare = c(0.5, 0.5)
)
```

Arguments

`Output`	The output returned from TimeSeriesDataPrepare()
`MetricSelection`	Choose from MAE, MSE, and MAPE
`TrainValidateShare`	The value returned from TimeSeriesPrepare()

Value

Time series data sets to pass onto auto modeling functions

Author(s)

Adrian Antico

See Also

Other Time Series Helper: `FinalBuildArfima()`, `FinalBuildArima()`, `FinalBuildETS()`, `FinalBuildNNET()`, `FinalBuildTBATS()`, `FinalBuildTSLM()`, `GenerateParameterGrids()`, `OptimizeArfima()`, `OptimizeArima()`, `OptimizeETS()`, `OptimizeNNET()`, `OptimizeTBATS()`, `OptimizeTSLM()`, `ParallelAutoARIMA()`, `ParallelAutoArfima()`, `ParallelAutoETS()`, `ParallelAutoNNET()`, `ParallelAutoTBATS()`, `PredictArima()`, `RL_Performance()`, `Regular_Performance()`, `StackedTimeSeriesEnsembleForecast()`, `TimeSeriesDataPrepare`
`WideTimeSeriesEnsembleForecast()`

Examples

```
## Not run:
ParallelAutoTSLM(
  MetricSelection = "MAE",
  Output = NULL,
  TrainValidateShare = c(0.50,0.50))

## End(Not run)
```

`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) 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: AutoLimeAid(), EvalPlot(), LimeModel(), RedYellowGreen(), threshOptim()

Examples

```
## Not run:
# Create fake data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.70, N = 10000000, Classification = FALSE)
data.table::setnames(data, "Independent_Variable2", "Predict")

# Build plot
Plot <- RemixAutoML::ParDepCalPlots(
  data,
  PredictionColName = "Predict",
  TargetColName = "Adrian",
  IndepVar = "Independent_Variable1",
  GraphType = "calibration",
  PercentileBucket = 0.20,
  FactLevels = 10,
  Function = function(x) mean(x, na.rm = TRUE))

## End(Not run)
```

Partial_DT_GDL_Feature_Engineering
A version of the DT_GDL function for creating the GDL features for a new set of records

---

Description

For scoring models in production that have > 1 grouping variables and for when you need > 1 record (or records per grouping variables) returned. This function is for generating lags and moving averages (along with lags and moving averages off of time between records), for a partial set of records in your data set, typical new records that become available for model scoring. Column names and ordering will be identical to the output from the corresponding DT_GDL_Feature_Engineering() function, which most likely was used to create features for model training.

Usage

```
Partial_DT_GDL_Feature_Engineering(
  data,
  lags = c(seq(1, 5, 1)),
  periods = c(3, 5, 10, 15, 20, 25),
  SDperiods = c(seq(5, 95, 5)),
  Skewperiods = c(seq(5, 95, 5)),
  Kurtperiods = c(seq(5, 95, 5)),
  Quantileperiods = c(seq(5, 95, 5)),
  statsFUNs = c("mean"),
  targets = c("Target"),
  groupingVars = NULL,
  sortDateName = NULL,
  timeDiffTarget = NULL,
  timeAgg = NULL,
  WindowingLag = 1,
  Type = "Lag",
  Timer = TRUE,
  SimpleImpute = TRUE,
  AscRowByGroup = "temp",
  RecordsKeep = 1,
  AscRowRemove = 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.
SDperiods	A numeric vector of Standard Deviation rolling statistics window sizes you want to utilize in the calculations.
Skewperiods	A numeric vector of Skewness rolling statistics window sizes you want to utilize in the calculations.

`Kurtperiods` A numeric vector of Kurtosis rolling statistics window sizes you want to utilize in the calculations.

`Quantileperiods` A numeric vector of Quantile rolling statistics window sizes you want to utilize in the calculations.

`statsFUNs` Select from the following c("mean","sd","skew","kurt","q5","q10","q15","q20","q25","q30","q35","q

`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 the smallest numbers being the records for scoring (typically the most current in time).

`RecordsKeep` List the row number of AscRowByGroup and those data points will be returned

`AscRowRemove` Set to TRUE to remove the AscRowByGroup column upon returning data.

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()`, `AutoHierarchicalFourier()`, `AutoLagRollStatsScoring()`, `AutoLagRollStats()`, `AutoTransformationCreate()`, `AutoTransformationScore()`, `AutoWord2VecModeler()`, `ContinuousTimeDataGenerator()`, `CreateCalendarVariables()`, `CreateHolidayVariables()`, `DT_GDL_Feature_Engineering()`, `DummifyDT()`, `H2oAutoencoder()`, `ModelDataPrep()`, `TimeSeriesFill()`

Examples

```
## Not run:
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 <- Partial_DT_GDL_Feature_Engineering(
  data,
  lags           = c(1:5),
  periods        = c(seq(10,50,10)),
  SDperiods       = c(seq(5, 95, 5)),
  Skewperiods     = c(seq(5, 95, 5)),
  Kurtperiods     = c(seq(5, 95, 5)),
  Quantileperiods = c(seq(5, 95, 5)),
  statsFUNs      = c("mean","sd", "skew",
    "kurt","q5","q95"),
  targets        = c("Target"),
  groupingVars   = NULL,
  sortDateName   = "DateTime",
  timeDiffTarget = c("Time_Gap"),
  timeAgg        = "days",
  WindowingLag   = 1,
  Type           = "Lag",
  Timer          = TRUE,
  SimpleImpute   = TRUE,
  AscRowByGroup  = "temp",
  RecordsKeep    = c(1,5,100,2500),
  AscRowRemove   = TRUE)

## End(Not run)
```

PredictArima *PredictArima to forecast an Arima() model from the stats package*

Description

PredictArima is a function to overwrite the s3 generic <code>getS3method('predict','Arima')</code>

Usage

```
PredictArima(
  object = Results,
  n.ahead = FCPeriods,
  newxreg = NULL,
  se.fit = TRUE
)
```

Arguments

`object`	Object that stores the output from Arima()
`n.ahead`	Number of forecast periods to forecast
`newxreg`	NULL by default. Forward looking independent variables as matrix type
`se.fit`	Set to FALSE to not return prediction intervals with the forecast

Author(s)

Adrian Antico

See Also

Other Time Series Helper: FinalBuildArfima(), FinalBuildArima(), FinalBuildETS(), FinalBuildNNET(), FinalBuildTBATS(), FinalBuildTSLM(), GenerateParameterGrids(), OptimizeArfima(), OptimizeArima(), OptimizeETS(), OptimizeNNET(), OptimizeTBATS(), OptimizeTSLM(), ParallelAutoARIMA(), ParallelAutoArfima(), ParallelAutoETS(), ParallelAutoNNET(), ParallelAutoTBATS(), ParallelAutoTSLM() RL_Performance(), Regular_Performance(), StackedTimeSeriesEnsembleForecast(), TimeSeriesDataPrepare WideTimeSeriesEnsembleForecast()

PrintToPDF *PrintToPDF*

Description

PrintToPDF

Usage

```
PrintToPDF(
  Path,
  OutputName,
  ObjectList = NULL,
  Tables = FALSE,
  MaxPages = 500,
  Title = "Model Output",
  Width = 7,
  Height = 7,
  Paper = "USr",
  BackgroundColor = "transparent",
  ForegroundColor = "black"
)
```

Arguments

`Path`	Path file to the location where you want your pdf saved
`OutputName`	Supply a name for the file you want saved
`ObjectList`	List of objects to print to pdf
`Tables`	TRUE for data tables, FALSE for plots
`MaxPages`	Default of 500
`Title`	The title of the pdf
`Width`	Default is 7
`Height`	Default is 7
`Paper`	'USr' for landscape. 'special' means that Width and Height are used to determine page size
`BackgroundColor`	Default is 'transparent'
`ForegroundColor`	Default is 'black'

Author(s)

Adrian Antico

See Also

Other Misc: AutoH2OTextPrepScoring(), ChartTheme(), RPM_Binomial_Bandit(), tokenizeH2O()

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()

Examples

```
## Not run:
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)

## End(Not run)
```

QA_WALMARTDATAGENERATOR

QA_WALMARTDATAGENERATOR

Description

QA_WALMARTDATAGENERATOR

Usage

```
QA_WALMARTDATAGENERATOR(data, Groups = 1L, TimeUnit__ = "WEEK")
```

Arguments

data	supply walmart data for either a single group or two group case. For no group, use XX
Groups	Supply either 0L, 1L, or 2L to indicate the number of group variables to have tested
TimeUnit__	= TimeUnit_

Author(s)

Adrian Antico

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: AutoLimeAid(), EvalPlot(), LimeModel(), ParDepCalPlots(), threshOptim()

Examples

```
## Not run:
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 = 0.01,
  Cores = 1,
  Boundaries = c(0.05,0.75))

## End(Not run)
```

`Regular_Performance` *Regular_Performance creates and stores model results in Experiment Grid*

Description

Regular_Performance creates and stores model results in Experiment Grid

Usage

```
Regular_Performance(
  Model = NULL,
  Results = Results,
  GridList = GridList,
  TrainValidateShare = c(0.5, 0.5),
  ExperimentGrid = ExperimentGrid,
```

```
    run = run,
    train = train,
    ValidationData = ValidationData,
    HoldOutPeriods = HoldOutPeriods
)
```

Arguments

Model	Set to ets, tbats, arfima, tslm, nnetar
Results	This is a time series model
GridList	List
TrainValidateShare	The values used to blend training and validation performance
ExperimentGrid	The results collection table
run	Iterator
train	Data set
ValidationData	Data set
HoldOutPeriods	Passthrough

Author(s)

Adrian Antico

See Also

Other Time Series Helper: FinalBuildArfima(), FinalBuildArima(), FinalBuildETS(), FinalBuildNNET(), FinalBuildTBATS(), FinalBuildTSLM(), GenerateParameterGrids(), OptimizeArfima(), OptimizeArima(), OptimizeETS(), OptimizeNNET(), OptimizeTBATS(), OptimizeTSLM(), ParallelAutoARIMA(), ParallelAutoArfima(), ParallelAutoETS(), ParallelAutoNNET(), ParallelAutoTBATS(), ParallelAutoTSLM(), PredictArima(), RL_Performance(), StackedTimeSeriesEnsembleForecast(), TimeSeriesDataPrepare(), WideTimeSeriesEnsembleForecast()

RemixClassificationMetrics

RemixClassificationMetrics

Description

RemixClassificationMetrics

Usage

```
RemixClassificationMetrics(
  MLModels = c("catboost", "h2oautoml", "h2ogbm", "h2odrf", "xgboost"),
  TargetVariable = "Value",
  Thresholds = seq(0.01, 0.99, 0.01),
  CostMatrix = c(1, 0, 0, 1),
  ClassLabels = c(1, 0),
  CatBoostTestData = NULL,
```

```
  H2oAutoMLTestData = NULL,
  H2oGBMTestData = NULL,
  H2oDRFTestData = NULL,
  H2oGLMTestData = NULL,
  XGBoostTestData = NULL
)
```

Arguments

`MLModels`	A vector of model names from remixautoml. e.g. c("catboost","h2oautoml","h2ogbm","h2odrf","h2o
`TargetVariable`	Name of your target variable
`Thresholds`	seq(0.01,0.99,0.01),
`CostMatrix`	c(1,0,0,1),
`ClassLabels`	c(1,0),
`CatBoostTestData`	Test data returned from AutoCatBoostClassifier
`H2oAutoMLTestData`	Test data returned from AutoCatBoostClassifier
`H2oGBMTestData`	Test data returned from AutoH2oGBMClassifier
`H2oDRFTestData`	Test data returned from AutoH2oDRFClassifier
`H2oGLMTestData`	Test data returned from AutoH2oGLMClassifier
`XGBoostTestData`	Test data returned from AutoXGBoostClassifier

Author(s)

Adrian Antico

See Also

Other Model Evaluation: `ClassificationMetrics()`, `DT_BinaryConfusionMatrix()`

Examples

```
## Not run:
RemixClassificationMetrics <- function(
  MLModels = c("catboost",
               "h2oautoml",
               "h2ogbm",
               "h2odrf",
               "xgboost"),
  TargetVariable = "Value",
  Thresholds = seq(0.01,0.99,0.01),
  CostMatrix = c(1,0,0,1),
  ClassLabels = c(1,0),
  CatBoostTestData = NULL,
  H2oAutoMLTestData = NULL,
  H2oGBMTestData = NULL,
  H2oDRFTestData = NULL,
  H2oGLMTestData = NULL,
  XGBoostTestData = NULL)

## End(Not run)
```

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 Graphics: TimeSeriesPlotter(), multiplot()

Examples

```
## Not run:
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()

## End(Not run)
```

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",
  Lags = 5,
  MA = 5,
  SLags = 0,
  SMA = 0,
  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
`Lags`	the largest lag or moving average (seasonal too) values for the arima fit
`MA`	Max moving average
`SLags`	Max seasonal lags
`SMA`	Max seasonal moving averages
`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(), H2oIsolationForest()

Examples

```
## Not run:
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",
  Lags = 5,
  MA = 5,
  SLags = 0,
  SMA = 0,
  tstat = 4)
data     <- stuff[[1]]
model    <- stuff[[2]]
outliers <- data[type != "<NA>"]

## End(Not run)
```

RL_Initialize *RL_Initialize*

Description

RL_Initialize sets up the components necessary for RL

Usage

```
RL_Initialize(
  ParameterGridSet = NULL,
  Alpha = 1L,
  Beta = 1L,
  SubDivisions = 1000L
)
```

Arguments

ParameterGridSet
: This is a list of tuning grids

Alpha
: Prior successes

Beta
: Prior trials

SubDivisions
: Tolerance for integration

Author(s)

Adrian Antico

See Also

Other Reinforcement Learning: RL_ML_Update(), RL_Update()

Examples

```
## Not run:
RL_Start <- RL_Initialize(
  ParameterGridSet = GridClusters,
  Alpha = Alpha,
  Beta = Beta,
  SubDivisions = 1000L)
BanditArmsN <- RL_Start[["BanditArmsN"]]
Successes <- RL_Start[["Successes"]]
Trials <- RL_Start[["Trials"]]
GridIDs <- RL_Start[["GridIDs"]]
BanditProbs <- RL_Start[["BanditProbs"]]

## End(Not run)
```

RL_ML_Update *RL_ML_Update*

Description

RL_ML_Update updates the bandit probabilities for selecting different grids

Usage

```
RL_ML_Update(
  ExperimentGrid = ExperimentGrid,
  ModelType = "classification",
  ModelRun = counter,
  NEWGrid = NewGrid,
  NewPerformance = NewPerformance,
  BestPerformance = BestPerformance,
  TrialVector = Trials,
  SuccessVector = Successes,
  GridIDS = GridIDs,
  BanditArmsCount = BanditArmsN,
```

```
    RunsWithoutNewWinner = RunsWithoutNewWinner,
    MaxRunsWithoutNewWinner = MaxRunsWithoutNewWinner,
    MaxNumberModels = MaxNumberModels,
    MaxRunMinutes = MaxRunMinutes,
    TotalRunTime = TotalRunTime,
    BanditProbabilities = BanditProbs
  )
```

Arguments

ExperimentGrid	This is a data.table of grid params and model results
ModelType	"classification", "regression", and "multiclass"
ModelRun	Model iteration number
NEWGrid	Previous grid passed in
NewPerformance	Internal
BestPerformance	Internal
TrialVector	Numeric vector with the total trials for each arm
SuccessVector	Numeric vector with the total successes for each arm
GridIDS	The numeric vector that identifies which grid is which
BanditArmsCount	The number of arms in the bandit
RunsWithoutNewWinner	Counter of the number of models previously built without being a new winner
MaxRunsWithoutNewWinner	Maximum number of models built without a new best model (constraint)
MaxNumberModels	Maximum number of models to build (constraint)
MaxRunMinutes	Run time constraint
TotalRunTime	Cumulative run time in minutes
BanditProbabilities	Inital probabilities from RL_Initialize()

Author(s)

Adrian Antico

See Also

Other Reinforcement Learning: RL_Initialize(), RL_Update()

Examples

```
## Not run:
RL_Update_Output <- RL_ML_Update(
  ExperimentGrid = ExperimentGrid,
  ModelRun = run,
  ModelType = "classification",
  NEWGrid = NewGrid,
  NewPerformance = NewPerformance,
  BestPerformance = BestPerformance,
```

```
    TrialVector = Trials,
    SuccessVector = Successes,
    GridIDS = GridIDs,
    BanditArmsCount = BanditArmsN,
    RunsWithoutNewWinner = RunsWithoutNewWinner,
    MaxRunsWithoutNewWinner = MaxRunsWithoutNewWinner,
    MaxNumberModels = MaxNumberModels,
    MaxRunMinutes = MaxRunMinutes,
    TotalRunTime = TotalRunTime,
    BanditProbabilities = BanditProbs)
  BanditProbs <- RL_Update_Output[["BanditProbs"]]
  Trials <- RL_Update_Output[["Trials"]]
  Successes <- RL_Update_Output[["Successes"]]
  NewGrid <- RL_Update_Output[["NewGrid"]]

## End(Not run)
```

RL_Performance *ARIMA_Performance creates and stores model results in Experiment Grid*

Description

ARIMA_Performance creates and stores model results in Experiment Grid

Usage

```
RL_Performance(
  Results = Results,
  NextGrid = NextGrid,
  TrainValidateShare = c(0.5, 0.5),
  MaxFourierTerms = NULL,
  XREGFC = XREGFC,
  ExperimentGrid = ExperimentGrid,
  run = run,
  train = train,
  ValidationData = ValidationData,
  HoldOutPeriods = HoldOutPeriods,
  FinalScore = FALSE
)
```

Arguments

`Results`	This is a time series model
`NextGrid`	Bandit grid
`TrainValidateShare`	The values used to blend training and validation performance
`MaxFourierTerms`	Numeric value
`XREGFC`	Fourier terms for forecasting
`ExperimentGrid`	The results collection table

run	Iterator
train	Data set
ValidationData	Data set
HoldOutPeriods	Passthrough
FinalScore	FALSE

Author(s)

Adrian Antico

See Also

Other Time Series Helper: FinalBuildArfima(), FinalBuildArima(), FinalBuildETS(), FinalBuildNNET(), FinalBuildTBATS(), FinalBuildTSLM(), GenerateParameterGrids(), OptimizeArfima(), OptimizeArima(), OptimizeETS(), OptimizeNNET(), OptimizeTBATS(), OptimizeTSLM(), ParallelAutoARIMA(), ParallelAutoArfima(), ParallelAutoETS(), ParallelAutoNNET(), ParallelAutoTBATS(), ParallelAutoTSLM(), PredictArima(), Regular_Performance(), StackedTimeSeriesEnsembleForecast(), TimeSeriesDataPrepare(), WideTimeSeriesEnsembleForecast()

RL_Update *RL_Update*

Description

RL_Update updates the bandit probabilities for selecting different grids

Usage

```
RL_Update(
  ExperimentGrid = ExperimentGrid,
  MetricSelection = MetricSelection,
  ModelRun = run,
  NEWGrid = NewGrid,
  TrialVector = Trials,
  SuccessVector = Successes,
  GridIDS = GridIDs,
  BanditArmsCount = BanditArmsN,
  RunsWithoutNewWinner = RunsWithoutNewWinner,
  MaxRunsWithoutNewWinner = MaxRunsWithoutNewWinner,
  MaxNumberModels = MaxNumberModels,
  MaxRunMinutes = MaxRunMinutes,
  TotalRunTime = TotalRunTime,
  BanditProbabilities = BanditProbs
)
```

Arguments

ExperimentGrid This is a data.table of grid params and model results

MetricSelection The chosen metric to evalute models

ModelRun Model iteration number

NEWGrid Previous grid passed in

TrialVector Numeric vector with the total trials for each arm

SuccessVector Numeric vector with the total successes for each arm

GridIDS The numeric vector that identifies which grid is which

BanditArmsCount The number of arms in the bandit

RunsWithoutNewWinner Counter of the number of models previously built without being a new winner

MaxRunsWithoutNewWinner Maximum number of models built without a new best model (constraint)

MaxNumberModels Maximum number of models to build (constraint)

MaxRunMinutes Run time constraint

TotalRunTime Cumulative run time in minutes

BanditProbabilities Inital probabilities from RL_Initialize()

Author(s)

Adrian Antico

See Also

Other Reinforcement Learning: RL_Initialize(), RL_ML_Update()

Examples

```
## Not run:
RL_Update_Output <- RL_Update(
  ExperimentGrid = ExperimentGrid,
  MetricSelection = MetricSelection,
  ModelRun = run,
  NEWGrid = NewGrid,
  TrialVector = Trials,
  SuccessVector = Successes,
  GridIDS = GridIDs,
  BanditArmsCount = BanditArmsN,
  RunsWithoutNewWinner = RunsWithoutNewWinner,
  MaxRunsWithoutNewWinner = MaxRunsWithoutNewWinner,
  MaxNumberModels = MaxNumberModels,
  MaxRunMinutes = MaxRunMinutes,
  TotalRunTime = TotalRunTime,
  BanditProbabilities = BanditProbs)
BanditProbs <- RL_Update_Output[["BanditProbs"]]
Trials <- RL_Update_Output[["Trials"]]
Successes <- RL_Update_Output[["Successes"]]
```

```
NewGrid <- RL_Update_Output[["NewGrid"]]

## End(Not run)
```

`RPM_Binomial_Bandit` *RPM_Binomial_Bandit*

Description

RPM_Binomial_Bandit computes randomized probability matching probabilities for each arm being best in a multi-armed bandit. Close cousin to Thomson Sampling.

Usage

```
RPM_Binomial_Bandit(
  Success,
  Trials,
  Alpha = 1L,
  Beta = 1L,
  SubDivisions = 1000L
)
```

Arguments

`Success`	Vector of successes. One slot per arm.
`Trials`	Vector of trials. One slot per arm.
`Alpha`	Prior parameter for success
`Beta`	Prior parameter for trials
`SubDivisions`	Default is 100L in the stats package. Changed it to 1000 for this function.

Value

Probability of each arm being the best arm compared to all other arms.

Author(s)

Adrian Antico

See Also

Other Misc: `AutoH2OTextPrepScoring()`, `ChartTheme()`, `PrintToPDF()`, `tokenizeH2O()`

SQL_ClearTable *SQL_ClearTable*

Description

SQL_ClearTable get data from a database

Usage

```
SQL_ClearTable(
  DBConnection,
  SQLTableName = "",
  CloseChannel = TRUE,
  Errors = TRUE
)
```

Arguments

DBConnection	RemixAutoML::SQL_Server_DBConnection()
SQLTableName	The SQL statement you want to run
CloseChannel	TRUE to close when done, FALSE to leave the channel open
Errors	Set to TRUE to halt, FALSE to return -1 in cases of errors

Author(s)

Adrian Antico

See Also

Other Data Wrangling: AutoDataDictionaries(), ColumnSubsetDataTable(), DataDisplayMeta(), FakeDataGenerator(), FullFactorialCatFeatures(), SQL_DropTable(), SQL_Query_Push(), SQL_Query(), SQL_SaveTable(), SQL_Server_DBConnection(), SQL_UpdateTable(), TimeSeriesMelt()

SQL_DropTable *SQL_DropTable*

Description

SQL_DropTable get data from a database

Usage

```
SQL_DropTable(
  DBConnection,
  SQLTableName = "",
  CloseChannel = TRUE,
  Errors = TRUE
)
```

Arguments

`DBConnection`	RemixAutoML::SQL_Server_DBConnection()
`SQLTableName`	The SQL statement you want to run
`CloseChannel`	TRUE to close when done, FALSE to leave the channel open
`Errors`	Set to TRUE to halt, FALSE to return -1 in cases of errors

Author(s)

Adrian Antico

See Also

Other Data Wrangling: `AutoDataDictionaries()`, `ColumnSubsetDataTable()`, `DataDisplayMeta()`, `FakeDataGenerator()`, `FullFactorialCatFeatures()`, `SQL_ClearTable()`, `SQL_Query_Push()`, `SQL_Query()`, `SQL_SaveTable()`, `SQL_Server_DBConnection()`, `SQL_UpdateTable()`, `TimeSeriesMelt()`

`SQL_Query` *SQL_Query*

Description

SQL_Query get data from a database

Usage

```
SQL_Query(
  DBConnection,
  Query,
  ASIS = FALSE,
  CloseChannel = TRUE,
  RowsPerBatch = 1024
)
```

Arguments

`DBConnection`	RemixAutoML::SQL_Server_DBConnection()
`Query`	The SQL statement you want to run
`ASIS`	Auto column typing
`CloseChannel`	TRUE to close when done, FALSE to leave the channel open
`RowsPerBatch`	Rows default is 1024

Author(s)

Adrian Antico

See Also

Other Data Wrangling: `AutoDataDictionaries()`, `ColumnSubsetDataTable()`, `DataDisplayMeta()`, `FakeDataGenerator()`, `FullFactorialCatFeatures()`, `SQL_ClearTable()`, `SQL_DropTable()`, `SQL_Query_Push()`, `SQL_SaveTable()`, `SQL_Server_DBConnection()`, `SQL_UpdateTable()`, `TimeSeriesMelt()`

SQL_Query_Push *SQL_Query*

Description

SQL_Query get data from a database

Usage

```
SQL_Query_Push(DBConnection, Query, CloseChannel = TRUE)
```

Arguments

DBConnection	RemixAutoML::SQL_Server_DBConnection()
Query	The SQL statement you want to run
CloseChannel	TRUE to close when done, FALSE to leave the channel open

Author(s)

Adrian Antico

See Also

Other Data Wrangling: AutoDataDictionaries(), ColumnSubsetDataTable(), DataDisplayMeta(), FakeDataGenerator(), FullFactorialCatFeatures(), SQL_ClearTable(), SQL_DropTable(), SQL_Query(), SQL_SaveTable(), SQL_Server_DBConnection(), SQL_UpdateTable(), TimeSeriesMelt()

SQL_SaveTable *SQL_SaveTable*

Description

SQL_SaveTable get data from a database

Usage

```
SQL_SaveTable(
  DataToPush,
  DBConnection,
  SQLTableName = "",
  RowNames = NULL,
  ColNames = TRUE,
  CloseChannel = TRUE,
  AppendData = FALSE,
  AddPK = TRUE,
  Safer = TRUE
)
```

Arguments

DataToPush	data to be sent to warehouse
DBConnection	RemixAutoML::SQL_Server_DBConnection()
SQLTableName	The SQL statement you want to run
RowNames	c("Segment","Date")
ColNames	Column names in first row
CloseChannel	TRUE to close when done, FALSE to leave the channel open
AppendData	TRUE or FALSE
AddPK	Add a PK column to table
Safer	TRUE

Author(s)

Adrian Antico

See Also

Other Data Wrangling: AutoDataDictionaries(), ColumnSubsetDataTable(), DataDisplayMeta(), FakeDataGenerator(), FullFactorialCatFeatures(), SQL_ClearTable(), SQL_DropTable(), SQL_Query_Push(), SQL_Query(), SQL_Server_DBConnection(), SQL_UpdateTable(), TimeSeriesMelt()

SQL_Server_DBConnection

SQL_Server_DBConnection

Description

SQL_Server_DBConnection is a function to return data dictionary data in table form

Usage

```
SQL_Server_DBConnection(DataBaseName = "", Server = "")
```

Arguments

DataBaseName	Name of the database
Server	Name of the server to use

Author(s)

Adrian Antico

See Also

Other Data Wrangling: AutoDataDictionaries(), ColumnSubsetDataTable(), DataDisplayMeta(), FakeDataGenerator(), FullFactorialCatFeatures(), SQL_ClearTable(), SQL_DropTable(), SQL_Query_Push(), SQL_Query(), SQL_SaveTable(), SQL_UpdateTable(), TimeSeriesMelt()

SQL_UpdateTable *SQL_UpdateTable*

Description

SQL_UpdateTable get data from a database

Usage

```
SQL_UpdateTable(
  DataToPush,
  DBConnection,
  SQLTableName = "",
  Index = NULL,
  CloseChannel = TRUE,
  Verbose = TRUE,
  Test = FALSE,
  NAString = "NA",
  Fast = TRUE
)
```

Arguments

`DataToPush`	Update data table in warehouse with new values
`DBConnection`	RemixAutoML::SQL_Server_DBConnection()
`SQLTableName`	The SQL statement you want to run
`Index`	Column name of index
`CloseChannel`	TRUE to close when done, FALSE to leave the channel open
`Verbose`	TRUE or FALSE
`Test`	Set to TRUE to see if what you plan to do will work
`NAString`	Supply character string to supply missing values
`Fast`	Set to TRUE to update table in one shot versus row by row

Author(s)

Adrian Antico

See Also

Other Data Wrangling: `AutoDataDictionaries()`, `ColumnSubsetDataTable()`, `DataDisplayMeta()`, `FakeDataGenerator()`, `FullFactorialCatFeatures()`, `SQL_ClearTable()`, `SQL_DropTable()`, `SQL_Query_Push()`, `SQL_Query()`, `SQL_SaveTable()`, `SQL_Server_DBConnection()`, `TimeSeriesMelt()`

StackedTimeSeriesEnsembleForecast

TimeSeriesEnsembleForecast

Description

TimeSeriesEnsembleForecast to generate forecasts and ensemble data

Usage

```
StackedTimeSeriesEnsembleForecast(
  TS_Models = c("arima", "tbats", "nnet"),
  ML_Methods = c("CatBoost", "XGBoost", "H2oGBM", "H2oDRF"),
  CalendarFeatures = TRUE,
  HolidayFeatures = TRUE,
  FourierFeatures = NULL,
  Path = "C:/Users/aantico/Documents/Package",
  TargetName = "Weekly_Sales",
  DateName = "Date",
  NTrees = 750,
  TaskType = "GPU",
  GridTune = FALSE,
  FCPeriods = 5,
  MaxNumberModels = 5
)
```

Arguments

TS_Models	Select which ts model forecasts to ensemble
ML_Methods	Select which models to build for the ensemble
CalendarFeatures	TRUE or FALSE
HolidayFeatures	TRUE or FALSE
FourierFeatures	Full set of fourier features for train and score
Path	The path to the folder where the ts forecasts are stored
TargetName	"Weekly_Sales"
DateName	"Date"
NTrees	Select the number of trees to utilize in ML models
TaskType	GPU or CPU
GridTune	Set to TRUE to grid tune the ML models
FCPeriods	Number of periods to forecast
MaxNumberModels	The number of models to try for each ML model

Author(s)

Adrian Antico

See Also

Other Time Series Helper: FinalBuildArfima(), FinalBuildArima(), FinalBuildETS(), FinalBuildNNET(), FinalBuildTBATS(), FinalBuildTSLM(), GenerateParameterGrids(), OptimizeArfima(), OptimizeArima(), OptimizeETS(), OptimizeNNET(), OptimizeTBATS(), OptimizeTSLM(), ParallelAutoARIMA(), ParallelAutoArfima(), ParallelAutoETS(), ParallelAutoNNET(), ParallelAutoTBATS(), ParallelAutoTSLM(), PredictArima(), RL_Performance(), Regular_Performance(), TimeSeriesDataPrepare(), WideTimeSeriesEnse

---

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,
  MinThresh = 0.001,
  MaxThresh = 0.999,
  ThresholdPrecision = 0.001
)
```

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
MinThresh	Minimum value to consider for model threshold
MaxThresh	Maximum value to consider for model threshold
ThresholdPrecision	Incrementing value in search

Value

Optimal threshold and corresponding utilities for the range of thresholds tested

Author(s)

Adrian Antico

See Also

Other Model Evaluation and Interpretation: AutoLimeAid(), EvalPlot(), LimeModel(), ParDepCalPlots(), RedYellowGreen()

Examples

```
## Not run:
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,
                    MinThresh = 0.001,
                    MaxThresh = 0.999,
                    ThresholdPrecision = 0.001)
optimalThreshold <- data$Thresholds
allResults <- data$EvaluationTable

## End(Not run)
```

`TimeSeriesDataPrepare` *TimeSeriesDataPrepare is a function that takes raw data and returns time series data*

Description

TimeSeriesDataPrepare is a function that takes raw data and returns the necessary time series data and objects for model building. It also fills any time gaps with zeros. Use this before you run any time series model functions.

Usage

```
TimeSeriesDataPrepare(
  data,
  TargetName,
  DateName,
  Lags,
  SeasonalLags,
  MovingAverages,
  SeasonalMovingAverages,
  TimeUnit,
  FCPeriods,
```

```
  HoldOutPeriods,
  TSClean = TRUE,
  ModelFreq = TRUE,
  FinalBuild = FALSE
)
```

Arguments

`data`	Source data.table for forecasting
`TargetName`	Name of your target variable
`DateName`	Name of your date variable
`Lags`	The max number of lags you want to test
`SeasonalLags`	The max number of seasonal lags you want to test
`MovingAverages`	The max number of moving average terms
`SeasonalMovingAverages`	The max number of seasonal moving average terms
`TimeUnit`	The level of aggregation your dataset comes in. Choices include: 1Min, 5Min, 10Min, 15Min, and 30Min, hour, day, week, month, quarter, year
`FCPeriods`	The number of forecast periods you want to have forecasted
`HoldOutPeriods`	The number of holdout samples to compare models against
`TSClean`	TRUE or FALSE. TRUE will kick off a time series cleaning operation. Outliers will be smoothed and imputation will be conducted.
`ModelFreq`	TRUE or FALSE. TRUE will enable a model-based time frequency calculation for an alternative frequency value to test models on.
`FinalBuild`	Set to TRUE to create data sets with full data

Value

Time series data sets to pass onto auto modeling functions

Author(s)

Adrian Antico

See Also

Other Time Series Helper: `FinalBuildArfima()`, `FinalBuildArima()`, `FinalBuildETS()`, `FinalBuildNNET()`, `FinalBuildTBATS()`, `FinalBuildTSLM()`, `GenerateParameterGrids()`, `OptimizeArfima()`, `OptimizeArima()`, `OptimizeETS()`, `OptimizeNNET()`, `OptimizeTBATS()`, `OptimizeTSLM()`, `ParallelAutoARIMA()`, `ParallelAutoArfima()`, `ParallelAutoETS()`, `ParallelAutoNNET()`, `ParallelAutoTBATS()`, `ParallelAutoTSLM()`, `PredictArima()`, `RL_Performance()`, `Regular_Performance()`, `StackedTimeSeriesEnsembleForecast()`, `WideTimeSeriesEnsembleForecast()`

Examples

```
## Not run:
data <- data.table::fread(
  file.path(PathNormalizer(
    "C:\\Users\\aantico\\Documents\\Package\\data"),
    "tsdata.csv"))
```

```
TimeSeriesDataPrepare(
  data = data,
  TargetName = "Weekly_Sales",
  DateName = "Date",
  Lags = 5,
  MovingAverages,
  SeasonalMovingAverages,
  SeasonalLags = 1,
  TimeUnit = "week",
  FCPeriods = 10,
  HoldOutPeriods = 10,
  TSClean = TRUE,
  ModelFreq = TRUE,
  FinalBuild = FALSE)

## End(Not run)
```

`TimeSeriesFill` *TimeSeriesFill For Completing Time Series Data*

Description

TimeSeriesFill For Completing Time Series Data For Single Series or Time Series by Group

Usage

```
TimeSeriesFill(
  data = data,
  DateColumnName = "Date",
  GroupVariables = NULL,
  TimeUnit = "days",
  FillType = "all"
)
```

Arguments

`data`	Supply your full series data set here
`DateColumnName`	Supply the name of your date column
`GroupVariables`	Supply the column names of your group variables. E.g. "Group" or c("Group1","Group2")
`TimeUnit`	Choose from "second", "minute", "hour", "day", "week", "month", "quarter", "year"
`FillType`	Choose from "all" or "inner". Only relevant for when you have GroupVariables. The "all" option will take the max date and the min date of the entire data set and fill according to those. The "inner" option will grab the max and min dates by group levels and fill each group level based on those.

Value

Returns a data table with missing time series records filled (currently just zeros)

See Also

Other Feature Engineering: AutoDataPartition(), AutoHierarchicalFourier(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), ContinuousTimeDataGenerator(), CreateCalendarVariables(), CreateHolidayVariables(), DT_GDL_Feature_Engineering(), DummifyDT(), H2oAutoencoder(), ModelDataPrep(), Partial_DT_GDL_Feature_

Examples

```
## Not run:
data <- TimeSeriesFill(
  data,
  DateColumnName = "Date",
  GroupVariables = "GroupVar",
  TimeUnit = "days",
  FillType = "inner")

## End(Not run)
```

TimeSeriesMelt *TimeSeriesMelt*

Description

TimeSeriesMelt

Usage

```
TimeSeriesMelt(
  data,
  TargetVariable = NULL,
  DateVariable = NULL,
  GroupVariables = NULL
)
```

Arguments

data	source data
TargetVariable	vector of target variable names
DateVariable	Name of date variable
GroupVariables	Vector of group variable names

Author(s)

Adrian Antico

See Also

Other Data Wrangling: AutoDataDictionaries(), ColumnSubsetDataTable(), DataDisplayMeta(), FakeDataGenerator(), FullFactorialCatFeatures(), SQL_ClearTable(), SQL_DropTable(), SQL_Query_Push(), SQL_Query(), SQL_SaveTable(), SQL_Server_DBConnection(), SQL_UpdateTable()

TimeSeriesPlotter *TimeSeriesPlotter*

Description

TimeSeriesPlotter is a function to plot single or multiple lines on a single plot

Usage

```
TimeSeriesPlotter(
  data = data,
  TargetVariable = "TargetVariableName",
  DateVariable = "DateVariableName",
  GroupVariables = "GroupVariableName",
  VLineDate = NULL,
  Aggregate = NULL,
  NumberGroupsDisplay = 5,
  LevelsToDisplay = NULL,
  OtherGroupLabel = "Other",
  DisplayOtherGroup = FALSE,
  TextSize = 12,
  LineWidth = 1,
  Color = "blue",
  XTickMarks = "1 year",
  Size = 12,
  AngleX = 35,
  AngleY = 0,
  ChartColor = "lightsteelblue1",
  BorderColor = "darkblue",
  TextColor = "darkblue",
  GridColor = "white",
  BackGroundColor = "gray95",
  LegendPosition = "bottom",
  LegendTextColor = "darkblue",
  LegendTextSize = 10,
  ForecastLineColor = "black",
  Forecast = FALSE,
  PredictionIntervals = FALSE,
  TS_ModelID = NULL,
  PredictionIntervalColorInner = "aquamarine1",
  PredictionIntervalColorOuter = "peachpuff1"
)
```

Arguments

`data`	Source data
`TargetVariable`	Target variable
`DateVariable`	Date variable
`GroupVariables`	Group variables
`VLineDate`	Date of last actual target value

`Aggregate` Choose from 'sum' or 'mean'

`NumberGroupsDisplay` Number of lines to display

`LevelsToDisplay` Value

`OtherGroupLabel` Label to call all other group levels

`DisplayOtherGroup` If TRUE, a line will be shown with all levels that fall into 'other' otherwise no line will be shown

`TextSize` Default 12

`LineWidth` Numeric value. Default is 1

`Color` Set to "blue", "red", etc

`XTickMarks` Number of tick marks on x-axis. "1 minute","15 minutes","30 minutes","1 hour","3 hour","6 hour","12 hour","1 day","3 day","1 week","2 week","1 month","3 month","6 month","1 year","2 year","5 year","10 year"

`Size` Size of text on plot

`AngleX` Angle of text on x axis

`AngleY` Angle of text on y axis

`ChartColor` Color of chart background

`BorderColor` Color of border

`TextColor` Text color

`GridColor` Grid color

`BackGroundColor` Background color

`LegendPosition` Legend position

`LegendTextColor` Text color

`LegendTextSize` Text size

`ForecastLineColor` Forecast line color

`Forecast` Set to TRUE to use forecast plots

`PredictionIntervals` Set to TRUE to plot prediction intervals

`TS_ModelID` Select a model from the list for forecasting viewer

`PredictionIntervalColorInner` Fills 20th to 80th percentiles

`PredictionIntervalColorOuter` Fills 5th to 20th and 80th to 95th percentiles

Author(s)

Adrian Antico

See Also

Other Graphics: `RemixTheme()`, `multiplot()`

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(), ChartTheme(), PrintToPDF(), RPM_Binomial_Bandit()

Examples

```
## Not run:
data <- tokenizeH2O(data = data[["StringColumn"]])

## End(Not run)
```

WideTimeSeriesEnsembleForecast
WideTimeSeriesEnsembleForecast

Description

WideTimeSeriesEnsembleForecast to generate forecasts and ensemble data

Usage

```
WideTimeSeriesEnsembleForecast(
  TS_Models = c("arima", "tbats", "nnet"),
  ML_Methods = c("CatBoost", "XGBoost", "H2oGBM", "H2oDRF"),
  Path = "C:/Users/aantico/Documents/Package",
  TargetName = "Weekly_Sales",
  DateName = "Date",
  NTrees = 750,
  TaskType = "GPU",
  GridTune = FALSE,
  MaxNumberModels = 5
)
```

Arguments

TS_Models	Select which ts model forecasts to ensemble
ML_Methods	Select which models to build for the ensemble
Path	The path to the folder where the ts forecasts are stored
TargetName	"Weekly_Sales"
DateName	"Date"
NTrees	Select the number of trees to utilize in ML models
TaskType	GPU or CPU
GridTune	Set to TRUE to grid tune the ML models
MaxNumberModels	The number of models to try for each ML model

Author(s)

Adrian Antico

See Also

Other Time Series Helper: FinalBuildArfima(), FinalBuildArima(), FinalBuildETS(), FinalBuildNNET(), FinalBuildTBATS(), FinalBuildTSLM(), GenerateParameterGrids(), OptimizeArfima(), OptimizeArima(), OptimizeETS(), OptimizeNNET(), OptimizeTBATS(), OptimizeTSLM(), ParallelAutoARIMA(), ParallelAutoArfima(), ParallelAutoETS(), ParallelAutoNNET(), ParallelAutoTBATS(), ParallelAutoTSLM(), PredictArima(), RL_Performance(), Regular_Performance(), StackedTimeSeriesEnsembleForecast(), TimeSeriesDataPrepare()

XGBoostClassifierParams

XGBoostClassifierParams

Description

XGBoostClassifierParams

Usage

```
XGBoostClassifierParams(
  counter = NULL,
  NThreads = -1L,
  BanditArmsN = NULL,
  eval_metric = NULL,
  task_type = NULL,
  model_path = NULL,
  NewGrid = NULL,
  Grid = NULL,
  ExperimentalGrid = NULL,
  GridClusters = NULL
)
```

Arguments

counter	Passthrough
NThreads	= -1L,
BanditArmsN	Passthrough
eval_metric	Passthrough
task_type	Passthrough
model_path	Passthrough
NewGrid	Passthrough
Grid	Passthrough
ExperimentalGrid	Passthrough
GridClusters	Passthrough

Author(s)

Adrian Antico

See Also

Other Supervised Learning: AutoH2OScoring(), CatBoostClassifierParams(), CatBoostMultiClassParams(), CatBoostParameterGrids(), CatBoostRegressionParams(), XGBoostMultiClassParams(), XGBoostParameterGr
XGBoostRegressionMetrics(), XGBoostRegressionParams()

XGBoostMultiClassParams *XGBoostMultiClassParams*

Description

XGBoostMultiClassParams

Usage

```
XGBoostMultiClassParams(
  counter = NULL,
  num_class = NULL,
  NThreads = -1L,
  BanditArmsN = NULL,
  eval_metric = NULL,
  task_type = NULL,
  model_path = NULL,
  NewGrid = NULL,
  Grid = NULL,
  ExperimentalGrid = NULL,
  GridClusters = NULL
)
```

Arguments

counter	Passthrough
num_class	NULL
NThreads	= -1L,
BanditArmsN	Passthrough
eval_metric	Passthrough
task_type	Passthrough
model_path	Passthrough
NewGrid	Passthrough
Grid	Passthrough
ExperimentalGrid	Passthrough
GridClusters	Passthrough

Author(s)

Adrian Antico

See Also

Other Supervised Learning: AutoH2OScoring(), CatBoostClassifierParams(), CatBoostMultiClassParams(), CatBoostParameterGrids(), CatBoostRegressionParams(), XGBoostClassifierParams(), XGBoostParameterGr
XGBoostRegressionMetrics(), XGBoostRegressionParams()

XGBoostParameterGrids *XGBoostParameterGrids*

Description

XGBoostParameterGrids

Usage

```
XGBoostParameterGrids(
  TaskType = "CPU",
  Shuffles = 1L,
  NTrees = seq(500L, 5000L, 500L),
  Depth = seq(4L, 16L, 2L),
  LearningRate = seq(0.05, 0.4, 0.05),
  MinChildWeight = seq(1, 10, 1),
  SubSample = seq(0.55, 1, 0.05),
  ColSampleByTree = seq(0.55, 1, 0.05)
)
```

Arguments

`TaskType`	"GPU" or "CPU"
`Shuffles`	The number of shuffles you want to apply to each grid
`NTrees`	seq(500L, 5000L, 500L)
`Depth`	seq(4L, 16L, 2L)
`LearningRate`	seq(0.05,0.40,0.05)
`MinChildWeight`	seq(1.0, 10.0, 1.0)
`SubSample`	seq(0.55, 1.0, 0.05)
`ColSampleByTree`	seq(0.55, 1.0, 0.05)

Value

A list containing data.table's with the parameters shuffled and ready to test in the bandit framework

Author(s)

Adrian Antico

See Also

Other Supervised Learning: `AutoH2OScoring()`, `CatBoostClassifierParams()`, `CatBoostMultiClassParams()`, `CatBoostParameterGrids()`, `CatBoostRegressionParams()`, `XGBoostClassifierParams()`, `XGBoostMultiClassP`
`XGBoostRegressionMetrics()`, `XGBoostRegressionParams()`

`XGBoostRegressionMetrics` *XGBoostRegressionMetrics*

Description

XGBoostRegressionMetrics

Usage

```
XGBoostRegressionMetrics(grid_eval_metric, MinVal, calibEval)
```

Arguments

`grid_eval_metric`	Passthrough
`MinVal`	= -1L,
`calibEval`	Passthrough

Author(s)

Adrian Antico

See Also

Other Supervised Learning: AutoH2OScoring(), CatBoostClassifierParams(), CatBoostMultiClassParams(), CatBoostParameterGrids(), CatBoostRegressionParams(), XGBoostClassifierParams(), XGBoostMultiClassP
XGBoostParameterGrids(), XGBoostRegressionParams()

XGBoostRegressionParams *XGBoostRegressionParams*

Description

XGBoostRegressionParams

Usage

```
XGBoostRegressionParams(
  counter = NULL,
  NThreads = -1L,
  BanditArmsN = NULL,
  objective = NULL,
  eval_metric = NULL,
  task_type = NULL,
  model_path = NULL,
  NewGrid = NULL,
  Grid = NULL,
  ExperimentalGrid = NULL,
  GridClusters = NULL
)
```

Arguments

counter	Passthrough
NThreads	= -1L,
BanditArmsN	Passthrough
objective	Passthrough
eval_metric	Passthrough
task_type	Passthrough
model_path	Passthrough
NewGrid	Passthrough
Grid	Passthrough
ExperimentalGrid	Passthrough
GridClusters	Passthrough

Author(s)

Adrian Antico

See Also

Other Supervised Learning: AutoH2OScoring(), CatBoostClassifierParams(), CatBoostMultiClassParams(), CatBoostParameterGrids(), CatBoostRegressionParams(), XGBoostClassifierParams(), XGBoostMultiClassP
XGBoostParameterGrids(), XGBoostRegressionMetrics()

Index