Package 'RemixAutoML'

December 19, 2021

Title Remix Automated Machine Learning

Version 0.6.0

Date 2021-10-26

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. Built 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, data.table, doParallel, foreach, lubridate, timeDate

Suggests knitr, rmarkdown, gridExtra

VignetteBuilder knitr

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:

1

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

AppsPlotting   *AppsPlotting*

Description

Simple shiny app for viewing time series plots with box plots or line plots. You can use up to 3 categorical variables to filter by and one additional variable to filter as a bonus. You'll need to have shiny, shinyWidgets, htmltools

Usage

```
AppsPlotting(
  data,
  XVariable = NULL,
  YVariable = NULL,
  DateName = NULL,
  GroupVariables = NULL,
  FilterVariable = NULL,
  ModelOutputList = NULL,
  HeaderColor = "black",
  AppWidth = 12L,
  LogoWidth = "1000px",
  LogoHeight = "100px",
  GroupVarsBoxColor = "navy",
  VarsBoxColor = "purple",
  FilterBoxColor = "blue",
  PlotBoxColor = "aqua",
  CreatePlotButtonColor = "primary",
  UpdatePlotButtonColor = "primary",
  ResetPlotButtonColor = "primary",
  Browser = FALSE,
  Docker = FALSE,
  UserName_Password_DT = NULL,
  Debug = FALSE
)
```

Arguments

data	Source data.table
XVariable	Starter column name for x-variable. Not needed if x-axis is the DateName column
YVariable	Starter column name for y-variable
DateName	Starter column name for date-variable
GroupVariables	Starter column name for group-variables
FilterVariable	Starter column name for filter-variable

```
ModelOutputList
                 ML output from RemixAutoML Auto___() SL functions
```
HeaderColor   'black', 'blue', 'purple', 'green', 'red', 'yellow'

AppWidth   Width of boxes

```
GroupVarsBoxColor
```
Choose from 'red', 'yellow', 'aqua', 'blue', 'light-blue', 'green', 'navy', 'teal', 'olive', 'lime', 'orange', 'fuchsia', 'purple', 'maroon', 'black'

VarsBoxColor   Choose from 'red', 'yellow', 'aqua', 'blue', 'light-blue', 'green', 'navy', 'teal', 'olive', 'lime', 'orange', 'fuchsia', 'purple', 'maroon', 'black'

FilterBoxColor   Choose from 'red', 'yellow', 'aqua', 'blue', 'light-blue', 'green', 'navy', 'teal', 'olive', 'lime', 'orange', 'fuchsia', 'purple', 'maroon', 'black'

PlotBoxColor   Choose from 'red', 'yellow', 'aqua', 'blue', 'light-blue', 'green', 'navy', 'teal', 'olive', 'lime', 'orange', 'fuchsia', 'purple', 'maroon', 'black'

```
CreatePlotButtonColor
                 Choose from 'default', 'primary', 'warning', 'danger', 'success', 'royal'
UpdatePlotButtonColor
                 Choose from 'default', 'primary', 'warning', 'danger', 'success', 'royal'
ResetPlotButtonColor
                 Choose from 'default', 'primary', 'warning', 'danger', 'success', 'royal'
```
Browser   FALSE

Docker   FALSE

```
UserName_Password_DT
```
NULL. In order to enforce authentication, supply a data.table with columns 'UserName' which contains the names of your users and 'Password' which contains the acceptable passwords. E.g. data.table::data.table(UserName = c('Adrian Antico', 'Guest'), Password = c('Password1', 'Password2')). Case sensitivity applies.

Debug   FALSE

Author(s)

Adrian Antico

See Also

Other GUI: [RunRemixAutoML](#)()

Examples

```
## Not run:
# Pull Data
data <- data.table::fread(system.file('tests/QA_DataSets/ThreeGroup-FC-Walmart-XREG3.csv', package = "RemixA
data[, Date := as.Date(Date)]

# Run App
RemixAutoML::AppsPlotting(
  data,
  XVariable = NULL,
  YVariable = 'XREG1',
  DateName = 'Date',
  GroupVariables = names(data)[seq_len(3L)],
```

```
      FilterVariable = 'XREG1',
      ModelOutputList = NULL,
      HeaderColor = 'black',
      AppWidth = 12L,
      LogoWidth = '1000px',
      LogoHeight = '100px',
      GroupVarsBoxColor = 'navy',
      VarsBoxColor = 'purple',
      FilterBoxColor = 'blue',
      PlotBoxColor = 'aqua',
      CreatePlotButtonColor = 'default',
      UpdatePlotButtonColor = 'default',
      ResetPlotButtonColor = 'default',
      Docker = FALSE,
      Browser = FALSE,
      UserName_Password_DT = NULL,
      Debug = FALSE)

   # XVariable = 'Date'
   # YVariable = 'XREG1'
   # DateName = 'Date'
   # GroupVariables = names(data)[seq_len(3L)]
   # FilterVariable = 'XREG1'
   # ModelOutputList = NULL
   # Debug = TRUE
   # HeaderColor = 'black'
   # AppWidth = 12L
   # LogoWidth = '1000px'
   # LogoHeight = '100px'
   # GroupVarsBoxColor = 'navy'
   # VarsBoxColor = 'purple'
   # FilterBoxColor = 'blue'
   # PlotBoxColor = 'aqua'
   # CreatePlotButtonColor = 'default'
   # UpdatePlotButtonColor = 'default'
   # ResetPlotButtonColor = 'default'
   # Docker = FALSE
   # Browser = FALSE
   # UserName_Password_DT = NULL
   # Debug = TRUE

   ## End(Not run)
```

ArgNullCheck	*ArgNullCheck*

Description

ArgNullCheck

Usage

```
ArgNullCheck(Input, InputID, Default)
```

Arguments

Input	input
InputID	inputId
Default	Default value

Author(s)

Adrian Antico

See Also

Other Shiny: ArgNullCheck2(), BlankRow(), DateInput(), GenerateEvaluationMetrics(), NumericInput(), PickerInput_GetLevels(), PickerInput(), PreparePlotData(), ReactiveLoadCSV(), ReturnParam(), StoreArgs(), TextInput(), observeEventLoad()

Examples

```
## Not run:
ArgValue <<- RemixAutoML::ArgNullCheck(Input = input, InputID = "TS_Value", Default = 2)

## End(Not run)
```

ArgNullCheck2 *ArgNullCheck2*

Description

ArgNullCheck2

Usage

```
ArgNullCheck2(Input, InputID, Default, Type = "numeric")
```

Arguments

Input	input
InputID	inputId
Default	Default value
Type	numeric character logical

Author(s)

Adrian Antico

See Also

Other Shiny: ArgNullCheck(), BlankRow(), DateInput(), GenerateEvaluationMetrics(), NumericInput(), PickerInput_GetLevels(), PickerInput(), PreparePlotData(), ReactiveLoadCSV(), ReturnParam(), StoreArgs(), TextInput(), observeEventLoad()

Examples

```
## Not run:
ArgValue <<- RemixAutoML::ArgNullCheck2(Input = input, InputID = "TS_Value", Default = 2, Type = "numeric")

## End(Not run)
```

AssignData	*Assign a data.table by name from an environment*

Description

Assign a data.table by name from an environment

Usage

```
AssignData(data, env = globalenv())
```

Arguments

env	an environment
df	character, name of the object

Value

the object

AutoArfima	*AutoArfima*

Description

AutoArfima 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

```
AutoArfima(
  data,
  FilePath = NULL,
  TargetVariableName,
  DateColumnName,
  TimeAggLevel = "week",
  EvaluationMetric = "MAE",
  NumHoldOutPeriods = 5L,
  NumFCPeriods = 5L,
  MaxLags = 5L,
  MaxMovingAverages = 5L,
  TrainWeighting = 0.5,
  MaxConsecutiveFails = 12L,
  MaxNumberModels = 100L,
  MaxRunTimeMinutes = 10L,
  NumberCores = max(1L, min(4L, parallel::detectCores() - 2L))
)
```

Arguments

data	Source data.table
FilePath	NULL to return nothing. Provide a file path to save the model and xregs if available
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 arfima
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	Default max(1L, min(4L, parallel::detectCores()-2L))

Author(s)

Adrian Antico

See Also

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

Examples

```
## Not run:
# Create fake data
data <- RemixAutoML::FakeDataGenerator(TimeSeries = TRUE, TimeSeriesTimeAgg = "days")

# Build model
Output <- RemixAutoML::AutoArfima(
  data,
  FilePath = NULL,
  TargetVariableName = "Weekly_Sales",
  DateColumnName = "Date",
  TimeAggLevel = "weeks",
  EvaluationMetric = "MAE",
  NumHoldOutPeriods = 5L,
  NumFCPeriods = 5L,
  MaxLags = 5L,
  MaxMovingAverages = 5L,
  TrainWeighting = 0.50,
  MaxConsecutiveFails = 12L,
  MaxNumberModels = 100L,
  MaxRunTimeMinutes = 10L,
  NumberCores = max(1L, min(4L, parallel::detectCores()-2L)))

# Output
Output$ForecastPlot
Output$Forecast
Output$PerformanceGrid

## End(Not run)
```

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,
  FilePath = NULL,
  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,
  NumberCores = max(1L, min(4L, parallel::detectCores() - 2L)),
  Debug = FALSE
)
```

Arguments

`data`	Source data.table
`FilePath`	NULL to return nothing. Provide a file path to save the model and xregs if available
`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

NumberCores   Default max(1L, min(4L, parallel::detectCores()-2L))

Debug   Set to TRUE to print some steps

Author(s)

Adrian Antico

See Also

Other Automated Time Series: `AutoArfima`(), `AutoBanditSarima`(), `AutoETS`(), `AutoTBATS`(), `AutoTS`()

Examples

```
## Not run:
# Create fake data
data <- RemixAutoML::FakeDataGenerator(TimeSeries = TRUE, TimeSeriesTimeAgg = "days")

# Build models
Output <- RemixAutoML::AutoBanditNNet(
  data = data,
  FilePath = NULL,
  TargetVariableName = "Weekly_Sales",
  DateColumnName = "Date",
  TimeAggLevel = "day",
  EvaluationMetric = "MAE",
  NumHoldOutPeriods = 5L,
  NumFCPeriods = 5L,
  MaxLags = 5L,
  MaxSeasonalLags = 1L,
  MaxFourierPairs = 2L,
  TrainWeighting = 0.50,
  MaxConsecutiveFails = 12L,
  MaxNumberModels = 100L,
  MaxRunTimeMinutes = 10L,
  NumberCores = max(1L, min(4L, parallel::detectCores()-2L)),
  Debug = FALSE)

# Output
Output$ForecastPlot
Output$Forecast
Output$PerformanceGrid

## End(Not run)
```

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,
  FilePath = NULL,
  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, min(4L, parallel::detectCores() - 2L)),
  DebugMode = FALSE
)
```

Arguments

data	Source data.table
FilePath	NULL to return nothing. Provide a file path to save the model and xregs if available

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	Default max(1L, min(4L, parallel::detectCores()-2L))
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: [AutoArfima()](), [AutoBanditNNet()](), [AutoETS()](), [AutoTBATS()](), [AutoTS()]()

Examples

```
## Not run:
# Create fake data
data <- RemixAutoML::FakeDataGenerator(TimeSeries = TRUE, TimeSeriesTimeAgg = "days")

# Build models
Output <- RemixAutoML::AutoBanditSarima(
  data = data,
  FilePath = NULL,
  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 Default max(1L, min(4L, parallel::detectCores()-2L)),
  DebugMode = FALSE)

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

## End(Not run)
```

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", "Log", "LogPlus1", "Sqrt", "Asin", "Logit"),
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("minute", "hour", "wday", "mday", "yday", "week", "isoweek",
   "month", "quarter", "year"),
HolidayVariable = c("USPublicHolidays", "EasterGroup", "ChristmasGroup",
   "OtherEcclesticalFeasts"),
HolidayLookback = NULL,
HolidayLags = 1L,
HolidayMovingAverages = 1L:2L,
TimeTrendVariable = FALSE,
ZeroPadSeries = NULL,
DataTruncate = FALSE,
SplitRatios = c(0.7, 0.2, 0.1),
PartitionType = "timeseries",
TaskType = "GPU",
NumGPU = 1,
DebugMode = FALSE,
Timer = TRUE,
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 = NULL,
    LearningRate = NULL,
    RandomStrength = 1,
    BorderCount = 254,
    Depth = 6,
    RSM = 1,
    BootStrapType = "Bayesian",
    GrowPolicy = "SymmetricTree",
    ModelSizeReg = 0.5,
    FeatureBorderType = "GreedyLogSum",
    SamplingUnit = "Group",
    SubSample = NULL,
    ScoreFunction = "Cosine",
    MinDataInLeaf = 1
)
```

Arguments

data	Supply your full series data set here
TimeWeights NonNegativePred	Supply a value that will be multiplied by he time trend value
	TRUE or FALSE
RoundPreds	Rounding predictions to an integer value. TRUE or FALSE. Defaults to FALSE
TrainOnFull TargetColumnName	Set to TRUE to train on full data
	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 Group-Variables 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 NumOfParDepPlots	NULL Or supply a path. Data saved will be called 'ModelID'_data.csv
	Supply a number for the number of partial dependence plots you want returned
TargetTransformation	TRUE or FALSE. If TRUE, select the methods in the Methods arg you want tested. The best one will be applied.
Methods	Choose from 'YeoJohnson', 'BoxCox', 'Asinh', 'Log', 'LogPlus1', 'Sqrt', 'Asin', or 'Logit'. If more than one is selected, the one with the best normalization pearson statistic will be used. Identity is automatically selected and compared.
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) or list('day' = c(1:10), 'weeks' = c(1:4))
MA_Periods	Select the periods for all moving average variables you want to create. E.g. c(1:5,52) or list('day' = c(2:10), 'weeks' = c(2:4))
SD_Periods	Select the periods for all moving standard deviation variables you want to create. E.g. c(1:5,52) or list('day' = c(2:10), 'weeks' = c(2:4))
Skew_Periods	Select the periods for all moving skewness variables you want to create. E.g. c(1:5,52) or list('day' = c(2:10), 'weeks' = c(2:4))
Kurt_Periods	Select the periods for all moving kurtosis variables you want to create. E.g. c(1:5,52) or list('day' = c(2:10), 'weeks' = c(2:4))
Quantile_Periods	
	Select the periods for all moving quantiles variables you want to create. E.g. c(1:5,52) or list('day' = c(2:10), 'weeks' = c(2:4))
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 'minute', 'hour', 'wday', 'mday', 'yday', 'week', 'isoweek', 'month', 'quarter', 'year'
HolidayVariable	
	NULL, or select from 'USPublicHolidays', 'EasterGroup', 'ChristmasGroup', 'OtherEcclesticalFeasts'
HolidayLookback	
	Number of days in range to compute number of holidays from a given date in the data. If NULL, the number of days are computed for you.
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	NULL to do nothing. Otherwise, set to 'maxmax', 'minmax', 'maxmin', 'minmin'. See `TimeSeriesFill` for explanations of each type
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
PartitionType	Select 'random' for random data partitioning 'timeseries' for partitioning by time frames
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.
DebugMode	Defaults to FALSE. Set to TRUE to get a print statement of each high level comment in function
Timer	Set to FALSE to turn off the updating print statements for progress
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
LearningRate	Defaults to NULL. Catboost will dynamically define this if L2_Leaf_Reg is NULL and RMSE is chosen (otherwise catboost will default it to 0.03). Then you can pull it out of the model object and pass it back in should you wish.
RandomStrength	Default is 1
BorderCount	Default is 254
Depth	Depth of catboost model
RSM	CPU only. If TaskType is GPU then RSM will not be used
BootStrapType	If NULL, then if TaskType is GPU then Bayesian will be used. If CPU then MVS will be used. If MVS is selected when TaskType is GPU, then BootStrapType will be switched to Bayesian
GrowPolicy	Default is SymmetricTree. Others include Lossguide and Depthwise
ModelSizeReg	Defaults to 0.5. Set to 0 to allow for bigger models. This is for models with high cardinality categorical features. Valuues greater than 0 will shrink the model and quality will decline but models won't be huge.
FeatureBorderType	
	Defaults to 'GreedyLogSum'. Other options include: Median, Uniform, UniformAndQuantiles, MaxLogSum, MinEntropy
SamplingUnit	Default is Group. Other option is Object. if GPU is selected, this will be turned off unless the loss_function is YetiRankPairWise
SubSample	Can use if BootStrapType is neither Bayesian nor No. Pass NULL to use Catboost default. Used for bagging.
ScoreFunction	Default is Cosine. CPU options are Cosine and L2. GPU options are Cosine, L2, NewtonL2, and NewtomCosine (not available for Lossguide)
MinDataInLeaf	Defaults to 1. Used if GrowPolicy is not SymmetricTree

Value

See examples

Author(s)

Adrian Antico

See Also

Other Automated Panel Data Forecasting: AutoCatBoostHurdleCARMA(), AutoCatBoostVectorCARMA(), AutoH2OCARMA(), AutoLightGBMCARMA(), AutoLightGBMHurdleCARMA(), AutoXGBoostCARMA(), AutoXGBoostHurdleCARMA()

Examples

```
## Not run:

# Set up your output file path for saving results as a .csv
Path <- 'C:/YourPathHere'

# Run on GPU or CPU (some options in the grid tuning force usage of CPU for some runs)
TaskType = 'GPU'

# Define number of CPU threads to allow data.table to utilize
data.table::setDTthreads(percent = max(1L, parallel::detectCores()-2L))

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

# Ensure series have no missing dates (also remove series with more than 25% missing values)
data <- RemixAutoML::TimeSeriesFill(
  data,
  DateColumnName = 'Date',
  GroupVariables = c('Store','Dept'),
  TimeUnit = 'weeks',
  FillType = 'maxmax',
  MaxMissingPercent = 0.25,
  SimpleImpute = TRUE)

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

# Remove IsHoliday column
data[, IsHoliday := NULL]

# Create xregs (this is the include the categorical variables instead of utilizing only the interaction of them)
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))]

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

```r
# Define values for SplitRatios and FCWindow Args
N1 <- data1[, .N, by = c('Store','Dept')][1L, N]
N2 <- xregs[, .N, by = c('Store','Dept')][1L, N]

# Setup Grid Tuning & Feature Tuning data.table using a cross join of vectors
Tuning <- data.table::CJ(
  TimeWeights = c('None',0.999),
  MaxTimeGroups = c('weeks','months'),
  TargetTransformation = c('TRUE','FALSE'),
  Difference = c('TRUE','FALSE'),
  HoldoutTrain = c(6,18),
  Langevin = c('TRUE','FALSE'),
  NTrees = c(2500,5000),
  Depth = c(6,9),
  RandomStrength = c(0.75,1),
  L2_Leaf_Reg = c(3.0,4.0),
  RSM = c(0.75,'NULL'),
  GrowPolicy = c('SymmetricTree','Lossguide','Depthwise'),
  BootStrapType = c('Bayesian','MVS','No'))

# Remove options that are not compatible with GPU (skip over this otherwise)
Tuning <- Tuning[Langevin == 'TRUE' | (Langevin == 'FALSE' & RSM == 'NULL' & BootStrapType %in% c('Bayesian','No

# Randomize order of Tuning data.table
Tuning <- Tuning[order(runif(.N))]

# Load grid results and remove rows that have already been tested
if(file.exists(file.path(Path, 'Walmart_CARMA_Metrics.csv'))) {
  Metrics <- data.table::fread(file.path(Path, 'Walmart_CARMA_Metrics.csv'))
  temp <- data.table::rbindlist(list(Metrics,Tuning), fill = TRUE)
  temp <- unique(temp, by = c(4:(ncol(temp)-1)))
  Tuning <- temp[is.na(RunTime)][, .SD, .SDcols = names(Tuning)]
  rm(Metrics,temp)
}

# Define the total number of runs
TotalRuns <- Tuning[,.N]

# Kick off feature + grid tuning
for(Run in seq_len(TotalRuns)) {

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

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

  # Timer start
  StartTime <- Sys.time()

  # Run carma system
  CatBoostResults <- 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 = 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(1 - Tuning[Run, HoldoutTrain] / N2, Tuning[Run, HoldoutTrain] / N2),
  PartitionType = 'random',
  FC_Periods = N2-N1,
  TaskType = TaskType,
  NumGPU = 1,
  Timer = TRUE,
  DebugMode = TRUE,

  # Target variable transformations
  TargetTransformation = as.logical(Tuning[Run, TargetTransformation]),
Methods = c('YeoJohnson', 'BoxCox', 'Asinh', 'Log', 'LogPlus1', 'Sqrt', 'Asin', 'Logit'),
  Difference = as.logical(Tuning[Run, Difference]),
  NonNegativePred = TRUE,
  RoundPreds = FALSE,

  # Calendar-related features
  CalendarVariables = c('week','wom','month','quarter'),
  HolidayVariable = c('USPublicHolidays'),
  HolidayLookback = NULL,
  HolidayLags = c(1,2,3),
  HolidayMovingAverages = c(2,3),

  # Lags, moving averages, and other rolling stats
Lags = if(Tuning[Run, MaxTimeGroups] == 'weeks') c(1,2,3,4,5,8,9,12,13,51,52,53) else if(Tuning[Run, MaxTim
MA_Periods = if(Tuning[Run, MaxTimeGroups] == 'weeks') c(2,3,4,5,8,9,12,13,51,52,53) else if(Tuning[Run, Ma
  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 = TRUE,
  ZeroPadSeries = NULL,
  DataTruncate = FALSE,

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

  # ML evaluation output
  PDFOutputPath = NULL,
```

```
    SaveDataPath = NULL,
    NumOfParDepPlots = 0L,

    # ML loss functions
    EvalMetric = 'RMSE',
    EvalMetricValue = 1,
    LossFunction = 'RMSE',
    LossFunctionValue = 1,

    # ML tuning args
    NTrees = Tuning[Run, NTrees],
    Depth = Tuning[Run, Depth],
    L2_Leaf_Reg = Tuning[Run, L2_Leaf_Reg],
    LearningRate = 0.03,
    Langevin = as.logical(Tuning[Run, Langevin]),
    DiffusionTemperature = 10000,
    RandomStrength = Tuning[Run, RandomStrength],
    BorderCount = 254,
    RSM = if(Tuning[Run, RSM] == 'NULL') NULL else as.numeric(Tuning[Run, RSM]),
    GrowPolicy = Tuning[Run, GrowPolicy],
    BootStrapType = Tuning[Run, BootStrapType],
    ModelSizeReg = 0.5,
    FeatureBorderType = 'GreedyLogSum',
    SamplingUnit = 'Group',
    SubSample = NULL,
    ScoreFunction = 'Cosine',
    MinDataInLeaf = 1)

# Timer End
EndTime <- Sys.time()

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

# Fill NAs with 'Total' for totals and subtotals
for(cols in c('Store','Dept')) Results[, eval(cols) := data.table::fifelse(is.na(get(cols)), 'Total', get(co

# 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)]

  # Collect metrics for Total (feel free to switch to something else or no filter at all)
  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 (not overwrite)
  data.table::fwrite(Metrics, file = file.path(Path, 'Walmart_CARMA_Metrics.csv'), append = TRUE)

  # Remove objects (clear space before new runs)
  rm(CatBoostResults, Results, temp, Weekly_MAE, Weekly_MAPE, Monthly_MAE, Monthly_MAPE)

  # Garbage collection because of GPU
  gc()
}

## End(Not run)
```

AutoCatBoostClassifier

AutoCatBoostClassifier

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(
  OutputSelection = c("Importances", "EvalPlots", "EvalMetrics", "Score_TrainData"),
  data = NULL,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  PrimaryDateColumn = NULL,
  WeightsColumnName = NULL,
  IDcols = NULL,
```

```
  TrainOnFull = FALSE,
  task_type = "GPU",
  NumGPUs = 1,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  SaveInfoToPDF = FALSE,
  ModelID = "FirstModel",
  model_path = NULL,
  metadata_path = NULL,
  EvalMetric = "MCC",
  LossFunction = "Logloss",
  grid_eval_metric = "MCC",
  ClassWeights = c(1, 1),
  CostMatrixWeights = c(1, 0, 0, 1),
  NumOfParDepPlots = 0L,
  PassInGrid = NULL,
  GridTune = FALSE,
  MaxModelsInGrid = 30L,
  MaxRunsWithoutNewWinner = 20L,
  MaxRunMinutes = 24L * 60L,
  BaselineComparison = "default",
  MetricPeriods = 10L,
  Trees = 50L,
  Depth = 6,
  LearningRate = NULL,
  L2_Leaf_Reg = 3,
  RandomStrength = 1,
  BorderCount = 128,
  RSM = NULL,
  BootStrapType = NULL,
  GrowPolicy = "SymmetricTree",
  langevin = FALSE,
  diffusion_temperature = 10000,
  model_size_reg = 0.5,
  feature_border_type = "GreedyLogSum",
  sampling_unit = "Object",
  subsample = NULL,
  score_function = "Cosine",
  min_data_in_leaf = 1,
  DebugMode = FALSE
)
```

Arguments

OutputSelection

You can select what type of output you want returned. Choose from c('Importances', 'EvalPlots', 'EvalMetrics', 'PDFs', 'Score_TrainData')

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

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

TestData   This is your holdout data set. Catboost using both training and validation data

in the training process so you should evaluate out of sample performance with this data set.

TargetColumnName

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

FeatureColNames

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

PrimaryDateColumn

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

WeightsColumnName

Supply a column name for your weights column. Leave NULL otherwise

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

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

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.

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

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

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.

EvalMetric   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'

LossFunction   Default is NULL. Select the loss function of choice. c('Logloss','CrossEntropy','Lq','PairLogit','Pair

grid_eval_metric

Case sensitive. I typically choose 'Utility' or 'MCC'. Choose from 'Utility', 'MCC', 'Acc', 'F1_Score', 'F2_Score', 'F0.5_Score', 'TPR', 'TNR', 'FNR', 'FPR', 'FDR', 'FOR', 'NPV', 'PPV', 'ThreatScore'

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.

CostMatrixWeights

A vector with 4 elements c(True Positive Cost, False Negative Cost, False Positive Cost, True Negative Cost). Default c(1,0,0,1)

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)

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

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')

langevin   TRUE or FALSE. TRUE enables

diffusion_temperature

> Default value is 10000

model_size_reg Defaults to 0.5. Set to 0 to allow for bigger models. This is for models with high cardinality categorical features. Valuues greater than 0 will shrink the model and quality will decline but models won't be huge.

feature_border_type

> Defaults to 'GreedyLogSum'. Other options include: Median, Uniform, UniformAndQuantiles, MaxLogSum, MinEntropy

sampling_unit   Default is Group. Other option is Object. if GPU is selected, this will be turned off unless the LossFunction is YetiRankPairWise

subsample   Default is NULL. Catboost will turn this into 0.66 for BootStrapTypes Poisson and Bernoulli. 0.80 for MVS. Doesn't apply to others.

score_function Default is Cosine. CPU options are Cosine and L2. GPU options are Cosine, L2, NewtonL2, and NewtomCosine (not available for Lossguide)

min_data_in_leaf

> Default is 1. Cannot be used with SymmetricTree is GrowPolicy

DebugMode   Set to TRUE to get a printout of which step the function is on. FALSE, otherwise

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()`, `AutoLightGBMClassifier()`, `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,
```

```
      TrainOnFull = FALSE,
      DebugMode = FALSE,

      # Metadata args
      OutputSelection = c('Score_TrainData', 'Importances', 'EvalPlots', 'EvalMetrics'),
      ModelID = 'Test_Model_1',
      model_path = normalizePath('./'),
      metadata_path = normalizePath('./'),
      SaveModelObjects = FALSE,
      ReturnModelObjects = TRUE,
      SaveInfoToPDF = FALSE,

      # Data args
      data = data,
      ValidationData = NULL,
      TestData = NULL,
      TargetColumnName = 'Adrian',
      FeatureColNames = names(data)[!names(data) %in%
         c('IDcol_1','IDcol_2','Adrian')],
      PrimaryDateColumn = NULL,
      WeightsColumnName = NULL,
      IDcols = c('IDcol_1','IDcol_2'),

      # Evaluation args
      ClassWeights = c(1L,1L),
      CostMatrixWeights = c(1,0,0,1),
      EvalMetric = 'AUC',
      grid_eval_metric = 'MCC',
      LossFunction = 'Logloss',
      MetricPeriods = 10L,
      NumOfParDepPlots = ncol(data)-1L-2L,

      # Grid tuning args
      PassInGrid = NULL,
      GridTune = FALSE,
      MaxModelsInGrid = 30L,
      MaxRunsWithoutNewWinner = 20L,
      MaxRunMinutes = 24L*60L,
      BaselineComparison = 'default',

      # ML args
      Trees = 1000,
      Depth = 9,
      LearningRate = NULL,
      L2_Leaf_Reg = NULL,
      model_size_reg = 0.5,
      langevin = FALSE,
      diffusion_temperature = 10000,
      RandomStrength = 1,
      BorderCount = 128,
      RSM = 1,
      BootStrapType = 'Bayesian',
      GrowPolicy = 'SymmetricTree',
      feature_border_type = 'GreedyLogSum',
      sampling_unit = 'Object',
      subsample = NULL,
      score_function = 'Cosine',
```

```
    min_data_in_leaf = 1)

  ## End(Not run)
```

AutoCatBoostFunnelCARMA

AutoCatBoostFunnelCARMA

Description

AutoCatBoostFunnelCARMA is a forecasting model for cohort funnel forecasting for grouped data or non-grouped data

Usage

```
AutoCatBoostFunnelCARMA(
  data,
  GroupVariables = NULL,
  BaseFunnelMeasure = NULL,
  ConversionMeasure = NULL,
  ConversionRateMeasure = NULL,
  CohortPeriodsVariable = NULL,
  CalendarDate = NULL,
  CohortDate = NULL,
  TruncateDate = NULL,
  PartitionRatios = c(0.7, 0.2, 0.1),
  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,
  DebugMode = FALSE,
  CalendarVariables = c("wday", "mday", "yday", "week", "isoweek", "month", "quarter",
    "year"),
  HolidayGroups = c("USPublicHolidays", "EasterGroup", "ChristmasGroup",
    "OtherEcclesticalFeasts"),
  HolidayLookback = NULL,
  CohortHolidayLags = c(1L, 2L, 7L),
  CohortHolidayMovingAverages = c(3L, 7L),
  CalendarHolidayLags = c(1L, 2L, 7L),
  CalendarHolidayMovingAverages = c(3L, 7L),
  ImputeRollStats = -0.001,
  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,
  TaskType = "CPU",
  NumGPUs = 1,
  EvaluationMetric = "RMSE",
  LossFunction = "RMSE",
  MetricPeriods = 50L,
  NumOfParDepPlots = 1L,
  Trees = 3000L,
  Depth = 8L,
  L2_Leaf_Reg = NULL,
  LearningRate = NULL,
  Langevin = FALSE,
  DiffusionTemperature = 10000,
  RandomStrength = 1,
  BorderCount = 254,
  RSM = NULL,
  GrowPolicy = "SymmetricTree",
  BootStrapType = "Bayesian",
  ModelSizeReg = 0.5,
  FeatureBorderType = "GreedyLogSum",
  SamplingUnit = "Group",
  SubSample = NULL,
  ScoreFunction = "Cosine",
  MinDataInLeaf = 1
)
```

Arguments

data   data object

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.

PartitionRatios

    Requires three values for train, validation, and test data sets

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.

DebugMode   Internal use

CalendarVariables

    "wday", "mday", "yday", "week", "isoweek", "month", "quarter", "year"

HolidayGroups   c("USPublicHolidays","EasterGroup","ChristmasGroup","OtherEcclesticalFeasts")

HolidayLookback

    Number of days in range to compute number of holidays from a given date in the data. If NULL, the number of days are computed for you.

CohortHolidayLags

    c(1L, 2L, 7L),

CohortHolidayMovingAverages

    c(3L, 7L),

CalendarHolidayLags
                c(1L, 2L, 7L),
CalendarHolidayMovingAverages
                = c(3L, 7L),
ImputeRollStats
                Constant value to fill NA after running AutoLagRollStats()

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

TaskType   "GPU" or "CPU" for catboost training

NumGPUs   Number of GPU's you would like to utilize

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'

MetricPeriods   Number of trees to build before the internal catboost eval step happens

NumOfParDepPlots
                Number of partial dependence plots to return

Trees   Select the number of trees you want to have built to train the model

Depth   Depth of catboost model

L2_Leaf_Reg   l2 reg parameter

LearningRate   Defaults to NULL. Catboost will dynamically define this if L2_Leaf_Reg is NULL and RMSE is chosen (otherwise catboost will default it to 0.03). Then you can pull it out of the model object and pass it back in should you wish.

Langevin   Enables the Stochastic Gradient Langevin Boosting mode. If TRUE and Task-Type == 'GPU' then TaskType will be converted to 'CPU'

DiffusionTemperature
                Default is 10000

RandomStrength Default is 1

BorderCount   Default is 254

RSM   CPU only. If TaskType is GPU then RSM will not be used

GrowPolicy   Default is SymmetricTree. Others include Lossguide and Depthwise

BootStrapType   If NULL, then if TaskType is GPU then Bayesian will be used. If CPU then MVS will be used. If MVS is selected when TaskType is GPU, then BootStrapType will be switched to Bayesian

ModelSizeReg   Defaults to 0.5. Set to 0 to allow for bigger models. This is for models with high cardinality categorical features. Valuues greater than 0 will shrink the model and quality will decline but models won't be huge.

FeatureBorderType
                Defaults to 'GreedyLogSum'. Other options include: Median, Uniform, UniformAndQuantiles, MaxLogSum, MinEntropy

SamplingUnit   Default is Group. Other option is Object. if GPU is selected, this will be turned off unless the loss_function is YetiRankPairWise

SubSample   Can use if BootStrapType is neither Bayesian nor No. Pass NULL to use Catboost default. Used for bagging.

ScoreFunction   Default is Cosine. CPU options are Cosine and L2. GPU options are Cosine, L2, NewtonL2, and NewtomCosine (not available for Lossguide)

MinDataInLeaf   Defaults to 1. Used if GrowPolicy is not SymmetricTree

Author(s)

Adrian Antico

See Also

Other Automated Funnel Data Forecasting: `AutoCatBoostFunnelCARMAScoring()`, `AutoLightGBMFunnelCARMAScoring()`, `AutoLightGBMFunnelCARMA()`, `AutoXGBoostFunnelCARMAScoring()`, `AutoXGBoostFunnelCARMA()`

Examples

```
## Not run:
# Create Fake Data
data <- RemixAutoML::FakeDataGenerator(ChainLadderData = TRUE)

# Subset data for training
ModelDataBase <- data[CalendarDateColumn < '2020-01-01' & CohortDateColumn < '2020-01-01']
ModelData <- data.table::copy(ModelDataBase)

# Train Funne Model
TestModel <- RemixAutoML::AutoCatBoostFunnelCARMA(

  # Data Arguments
  data = ModelData,
  GroupVariables = NULL,
 BaseFunnelMeasure = "Leads", # if you have XREGS, supply vector such as c("Leads", "XREGS1", "XREGS2")
  ConversionMeasure = "Appointments",
  ConversionRateMeasure = NULL,
  CohortPeriodsVariable = "CohortDays",
  CalendarDate = "CalendarDateColumn",
  CohortDate = "CohortDateColumn",
  PartitionRatios = c(0.70,0.20,0.10),
  TruncateDate = NULL,
  TimeUnit = "days",
  TransformTargetVariable = TRUE,
  TransformMethods = c("Asinh","Asin","Log","LogPlus1","Sqrt","Logit"),
  AnomalyDetection = list(tstat_high = 3, tstat_low = -2),

  # MetaData Arguments
  Jobs = c("eval","train"),
  SaveModelObjects = FALSE,
  ModelID = "ModelTest",
  ModelPath = getwd(),
  MetaDataPath = NULL,
  DebugMode = TRUE,
  NumOfParDepPlots = 1L,

  # Feature Engineering Arguments
  CalendarTimeGroups = c("days","weeks","months"),
  CohortTimeGroups = c("days", "weeks"),
  CalendarVariables = c("wday","mday","yday","week","month","quarter","year"),
 HolidayGroups = c("USPublicHolidays","EasterGroup","ChristmasGroup","OtherEcclesticalFeasts"),
  HolidayLookback = NULL,
  CohortHolidayLags = c(1L,2L,7L),
  CohortHolidayMovingAverages = c(3L,7L),
  CalendarHolidayLags = c(1L,2L,7L),
  CalendarHolidayMovingAverages = c(3L,7L),
```

```
      # Time Series Features
      ImputeRollStats = -0.001,
      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,12
      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",

      # ML Grid Tuning
      PassInGrid = NULL,
      GridTune = FALSE,
      BaselineComparison = "default",
      MaxModelsInGrid = 25L,
      MaxRunMinutes = 180L,
      MaxRunsWithoutNewWinner = 10L,

      # ML Setup Parameters
      MetricPeriods = 10,
      LossFunction = 'MAE',
      EvaluationMetric = 'MAE',
      TaskType = "CPU",
      NumGPUs = 1,

      # ML Parameters
      Trees = 3000L,
      Depth = 8L,
      L2_Leaf_Reg = NULL,
      LearningRate = NULL,
      Langevin = FALSE,
      DiffusionTemperature = 10000,
      RandomStrength = 1,
      BorderCount = 254,
      RSM = NULL,
      GrowPolicy = "SymmetricTree",
      BootStrapType = "Bayesian",
      ModelSizeReg = 0.5,
      FeatureBorderType = "GreedyLogSum",
      SamplingUnit = "Group",
      SubSample = NULL,
      ScoreFunction = "Cosine",
      MinDataInLeaf = 1)

  # Separate out the Base Funnel Measures Data
  LeadsData <- data[, lapply(.SD, data.table::first), .SDcols = c("Leads"), by = c("CalendarDateColumn")]
  ModelData <- ModelDataBase[, Leads := NULL]

  # Forecast Funnel Model
```

```
Test <- RemixAutoML::AutoCatBoostFunnelCARMAScoring(
  TrainData = ModelData,
  ForwardLookingData = LeadsData,
  TrainEndDate = ModelData[, max(CalendarDateColumn)],
  ForecastEndDate = LeadsData[, max(CalendarDateColumn)],
  TrainOutput = TestModel$ModelOutput,
  ArgsList = TestModel$ArgsList,
  ModelPath = NULL,
  MaxCohortPeriod = 15,
  DebugMode = TRUE)

## End(Not run)
```

AutoCatBoostFunnelCARMAScoring
                            AutoCatBoostFunnelCARMAScoring

Description

AutoCatBoostFunnelCARMAScoring for generating forecasts

Usage

```
AutoCatBoostFunnelCARMAScoring(
  TrainData,
  ForwardLookingData = NULL,
  TrainEndDate = NULL,
  ForecastEndDate = NULL,
  ArgsList = NULL,
  TrainOutput = NULL,
  ModelPath = NULL,
  MaxCohortPeriod = NULL,
  DebugMode = FALSE
)
```

Arguments

TrainData	Data utilized in training. Do not put the BaseFunnelMeasure in this data set. Put it in the ForwardLookingData object
ForwardLookingData	
	Base funnel measure data. Needs to cover the span of the forecast horizon
TrainEndDate	Max date from the training data
ForecastEndDate	
	Max date to forecast out to
ArgsList	Output list from AutoCatBoostFunnelCARMA
TrainOutput	Pass in the model object to speed up forecasting
ModelPath	Path to model location
MaxCohortPeriod	
	Max cohort periods to utilize when forecasting
DebugMode	For debugging issues

Author(s)

Adrian Antico

See Also

Other Automated Funnel Data Forecasting: `AutoCatBoostFunnelCARMA()`, `AutoLightGBMFunnelCARMAScoring()`, `AutoLightGBMFunnelCARMA()`, `AutoXGBoostFunnelCARMAScoring()`, `AutoXGBoostFunnelCARMA()`

Examples

```
## Not run:
data <- RemixAutoML::FakeDataGenerator(ChainLadderData = TRUE)

# Subset data for training
ModelDataBase <- data[CalendarDateColumn < '2020-01-01' & CohortDateColumn < '2020-01-01']
ModelData <- data.table::copy(ModelDataBase)

# Train Funne Model
TestModel <- RemixAutoML::AutoCatBoostFunnelCARMA(

  # Data Arguments
  data = ModelData,
  GroupVariables = NULL,
 BaseFunnelMeasure = "Leads", # if you have XREGS, supply vector such as c("Leads", "XREGS1", "XREGS2")
  ConversionMeasure = "Appointments",
  ConversionRateMeasure = NULL,
  CohortPeriodsVariable = "CohortDays",
  CalendarDate = "CalendarDateColumn",
  CohortDate = "CohortDateColumn",
  PartitionRatios = c(0.70,0.20,0.10),
  TruncateDate = NULL,
  TimeUnit = "days",
  TransformTargetVariable = TRUE,
  TransformMethods = c("Asinh","Asin","Log","LogPlus1","Sqrt","Logit"),
  AnomalyDetection = list(tstat_high = 3, tstat_low = -2),

  # MetaData Arguments
  Jobs = c("eval","train"),
  SaveModelObjects = FALSE,
  ModelID = "ModelTest",
  ModelPath = getwd(),
  MetaDataPath = NULL,
  DebugMode = TRUE,
  NumOfParDepPlots = 1L,

  # Feature Engineering Arguments
  CalendarTimeGroups = c("days","weeks","months"),
  CohortTimeGroups = c("days", "weeks"),
  CalendarVariables = c("wday","mday","yday","week","month","quarter","year"),
 HolidayGroups = c("USPublicHolidays","EasterGroup","ChristmasGroup","OtherEcclesticalFeasts"),
  HolidayLookback = NULL,
  CohortHolidayLags = c(1L,2L,7L),
  CohortHolidayMovingAverages = c(3L,7L),
  CalendarHolidayLags = c(1L,2L,7L),
  CalendarHolidayMovingAverages = c(3L,7L),
```

```
 # Time Series Features
 ImputeRollStats = -0.001,
 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,12
 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",

 # ML Grid Tuning
 PassInGrid = NULL,
 GridTune = FALSE,
 BaselineComparison = "default",
 MaxModelsInGrid = 25L,
 MaxRunMinutes = 180L,
 MaxRunsWithoutNewWinner = 10L,

 # ML Setup Parameters
 MetricPeriods = 10,
 LossFunction = 'MAE',
 EvaluationMetric = 'MAE',
 TaskType = "CPU",
 NumGPUs = 1,

 # ML Parameters
 Trees = 3000L,
 Depth = 8L,
 L2_Leaf_Reg = NULL,
 LearningRate = NULL,
 Langevin = FALSE,
 DiffusionTemperature = 10000,
 RandomStrength = 1,
 BorderCount = 254,
 RSM = NULL,
 GrowPolicy = "SymmetricTree",
 BootStrapType = "Bayesian",
 ModelSizeReg = 0.5,
 FeatureBorderType = "GreedyLogSum",
 SamplingUnit = "Group",
 SubSample = NULL,
 ScoreFunction = "Cosine",
 MinDataInLeaf = 1)

# Separate out the Base Funnel Measures Data
LeadsData <- data[, lapply(.SD, data.table::first), .SDcols = c("Leads"), by = c("CalendarDateColumn")]
ModelData <- ModelDataBase[, Leads := NULL]

# Forecast Funnel Model
Test <- RemixAutoML::AutoCatBoostFunnelCARMAScoring(
```

```
  TrainData = ModelData,
  ForwardLookingData = LeadsData,
  TrainEndDate = ModelData[, max(CalendarDateColumn)],
  ForecastEndDate = LeadsData[, max(CalendarDateColumn)],
  TrainOutput = TestModel$ModelOutput,
  ArgsList = TestModel$ArgsList,
  ModelPath = NULL,
  MaxCohortPeriod = 15,
  DebugMode = 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,
  TimeWeights = 1,
  FC_Periods = 30,
  TimeUnit = "week",
  TimeGroups = c("weeks", "months"),
  NumOfParDepPlots = 10L,
  TargetTransformation = FALSE,
  Methods = c("BoxCox", "Asinh", "Log", "LogPlus1", "Sqrt", "Asin", "Logit"),
  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",
      "wom", "isoweek", "month", "quarter", "year"),
    HolidayVariable = c("USPublicHolidays", "EasterGroup", "ChristmasGroup",
      "OtherEcclesticalFeasts"),
    HolidayLookback = NULL,
    HolidayLags = 1L,
    HolidayMovingAverages = 1L:2L,
    TimeTrendVariable = FALSE,
    ZeroPadSeries = NULL,
    DataTruncate = FALSE,
    SplitRatios = c(0.7, 0.2, 0.1),
    PartitionType = "timeseries",
    Timer = TRUE,
    DebugMode = FALSE,
    TaskType = "GPU",
    NumGPU = 1,
    EvalMetric = "RMSE",
    GridTune = FALSE,
    PassInGrid = NULL,
    ModelCount = 100,
    MaxRunsWithoutNewWinner = 50,
    MaxRunMinutes = 24L * 60L,
    NTrees = list(classifier = 200, regression = 200),
    Depth = list(classifier = 9, regression = 9),
    LearningRate = list(classifier = NULL, regression = NULL),
    L2_Leaf_Reg = list(classifier = NULL, regression = NULL),
    RandomStrength = list(classifier = 1, regression = 1),
    BorderCount = list(classifier = 254, regression = 254),
    BootStrapType = list(classifier = "Bayesian", regression = "Bayesian")
  )
```

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.

TimeWeights	Timeweights creation. Supply a value, such as 0.9999
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	Choose from 'YeoJohnson', 'BoxCox', 'Asinh', 'Log', 'LogPlus1', 'Sqrt', 'Asin', or 'Logit'. If more than one is selected, the one with the best normalization pearson statistic will be used. Identity is automatically selected and compared.
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'
HolidayLookback	
	Number of days in range to compute number of holidays from a given date in the data. If NULL, the number of days are computed for you.

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
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
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

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()`, `AutoLightGBMCARMA()`, `AutoLightGBMHurdleCARMA()`, `AutoXGBoostCARMA()`, `AutoXGBoostHurdleCARMA()`

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'),
  TimeWeights = 1,
  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', 'Sqrt', 'Logit', 'YeoJohnson'),
```

```
      Difference = FALSE,
      NonNegativePred = FALSE,
      RoundPreds = FALSE,

      # Date features
      CalendarVariables = c('week', 'wom', 'month', 'quarter'),
      HolidayVariable = c('USPublicHolidays',
        'EasterGroup',
        'ChristmasGroup','OtherEcclesticalFeasts'),
      HolidayLookback = NULL,
      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'),
  TimeWeights = 1,
  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', 'Sqrt', 'Logit', 'YeoJohnson'),
  Difference = FALSE,
  NonNegativePred = FALSE,
  Threshold = NULL,
  RoundPreds = FALSE,

  # Date features
  CalendarVariables = c('week', 'wom', 'month', 'quarter'),
  HolidayVariable = c('USPublicHolidays',
                      'EasterGroup',
                      'ChristmasGroup','OtherEcclesticalFeasts'),
  HolidayLookback = NULL,
```

```
    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' = 200, 'regression' = 200),
    Depth = list('classifier' = 9, 'regression' = 9),
    LearningRate = list('classifier' = NULL, 'regression' = NULL),
    L2_Leaf_Reg = list('classifier' = NULL, 'regression' = NULL),
    RandomStrength = list('classifier' = 1, 'regression' = 1),
    BorderCount = list('classifier' = 254, 'regression' = 254),
    BootStrapType = list('classifier' = 'Bayesian', 'regression' = 'Bayesian'))

## End(Not run)
```

AutoCatBoostHurdleModel

AutoCatBoostHurdleModel

Description

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

Usage

```
AutoCatBoostHurdleModel(
```

```
      data = NULL,
      TrainOnFull = FALSE,
      ValidationData = NULL,
      TestData = NULL,
      Buckets = 0L,
      TargetColumnName = NULL,
      FeatureColNames = NULL,
      PrimaryDateColumn = NULL,
      WeightsColumnName = NULL,
      IDcols = NULL,
      TransformNumericColumns = NULL,
      Methods = c("Asinh", "Asin", "Log", "LogPlus1", "Sqrt", "Logit"),
      ClassWeights = NULL,
      SplitRatios = c(0.7, 0.2, 0.1),
      task_type = "GPU",
      ModelID = "ModelTest",
      Paths = NULL,
      DebugMode = FALSE,
      MetaDataPaths = NULL,
      SaveModelObjects = FALSE,
      ReturnModelObjects = TRUE,
      NumOfParDepPlots = 10L,
      PassInGrid = NULL,
      GridTune = FALSE,
      BaselineComparison = "default",
      MaxModelsInGrid = 1L,
      MaxRunsWithoutNewWinner = 20L,
      MaxRunMinutes = 60L * 60L,
      MetricPeriods = 25L,
      Langevin = FALSE,
      DiffusionTemperature = 10000,
      Trees = list(classifier = 500, regression = 500),
      Depth = list(classifier = 8, regression = 8),
      RandomStrength = list(classifier = 1, regression = 1),
      BorderCount = list(classifier = 254, regression = 254),
      LearningRate = list(classifier = NULL, regression = NULL),
      L2_Leaf_Reg = list(classifier = NULL, regression = NULL),
      RSM = list(classifier = 1, regression = 1),
      BootStrapType = list(classifier = "Bayesian", regression = "Bayesian"),
      GrowPolicy = list(classifier = "SymmetricTree", regression = "SymmetricTree")
  )
```

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 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 Primary-
                           DateColumn)

PrimaryDateColumn
                           Supply a date column if the data is functionally related to it

WeightsColumnName
                           Column name for weights variable

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

DebugMode   Print steps to screen by setting to TRUE

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,

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')
Shuffles         = 2L,
```

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 - Hurdle Modeling: `AutoH2oDRFHurdleModel`(), `AutoH2oGBMHurdleModel`(), `AutoLightGBMHurdleModel`(), `AutoXGBoostHurdleModel`()

Examples

```
## Not run:
# Test data.table
CatBoost_QA <- data.table::CJ(
  TOF = c(TRUE,FALSE),
  Classification = c(TRUE,FALSE),
  TaskType = c("CPU","GPU"),
  Success = "Failure",
  PartitionInFunction = c(TRUE,FALSE), sorted = FALSE
)

# Remove impossible combinations
CatBoost_QA <- CatBoost_QA[!(PartitionInFunction & TOF)]
CatBoost_QA[, RunNumber := seq_len(.N)]


# Path File
Path <- getwd()

#        TOF Classification TaskType Success PartitionInFunction RunNumber
# 1:    TRUE           TRUE      CPU Failure               FALSE         1 success
# 2:    TRUE           TRUE      GPU Failure               FALSE         2 success
# 3:    TRUE          FALSE      CPU Failure               FALSE         3 success
# 4:    TRUE          FALSE      GPU Failure               FALSE         4 success
# 5:   FALSE           TRUE      CPU Failure                TRUE         5 fail
# 6:   FALSE           TRUE      CPU Failure               FALSE         6 fail
```

```
# 7:  FALSE            TRUE      GPU Failure                  TRUE        7  fail
# 8:  FALSE            TRUE      GPU Failure                  FALSE       8  fail
# 9:  FALSE            FALSE     CPU Failure                  TRUE        9  fail
# 10: FALSE            FALSE     CPU Failure                  FALSE      10  fail
# 11: FALSE            FALSE     GPU Failure                  TRUE       11  fail
# 12: FALSE            FALSE     GPU Failure                  FALSE      12  fail

# AutoCatBoostHurdleModel
# run = 1
# run = 2
for(run in seq_len(CatBoost_QA[,.N])) {

  # Define values
  tasktypemode <- CatBoost_QA[run, TaskType]
  tof <- CatBoost_QA[run, TOF]
  PartitionInFunction <- CatBoost_QA[run, PartitionInFunction]
  Classify <- CatBoost_QA[run, Classification]
  Tar <- "Adrian"

  # Get data
  if(Classify) {
    data <- RemixAutoML::FakeDataGenerator(N = 15000, ZIP = 1)
  } else {
    data <- RemixAutoML::FakeDataGenerator(N = 15000, ZIP = 2)
  }

  # Partition Data
  if(!tof && !PartitionInFunction) {
    Sets <- RemixAutoML::AutoDataPartition(
      data = data,
      NumDataSets = 3,
      Ratios = c(0.7,0.2,0.1),
      PartitionType = "random",
      StratifyColumnNames = "Adrian",
      TimeColumnName = NULL)
    TTrainData <- Sets$TrainData
    VValidationData <- Sets$ValidationData
    TTestData <- Sets$TestData
    rm(Sets)
  } else {
    TTrainData <- data.table::copy(data)
    VValidationData <- NULL
    TTestData <- NULL
  }

  # Run function
  TestModel <- tryCatch({RemixAutoML::AutoCatBoostHurdleModel(

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

    # Data related args
    data = TTrainData,
    ValidationData = VValidationData,
```

```
    TestData = TTestData,
    WeightsColumnName = NULL,
    TrainOnFull = tof,
    Buckets = if(Classify) 0L else c(0,2,3),
    TargetColumnName = "Adrian",
  FeatureColNames = names(TTrainData)[!names(data) %in% c("Adrian","IDcol_1","IDcol_2","IDcol_3","IDcol_4",
    PrimaryDateColumn = "DateTime",
    IDcols = c("IDcol_1","IDcol_2","IDcol_3","IDcol_4","IDcol_5","DateTime"),
    DebugMode = TRUE,

    # Metadata args
    Paths = Path,
    MetaDataPaths = Path,
    TransformNumericColumns = NULL,
    Methods = c('Asinh', 'Asin', 'Log', 'LogPlus1', 'Sqrt', 'Logit'),
    ClassWeights = NULL,
    SplitRatios = if(PartitionInFunction) c(0.70, 0.20, 0.10) else NULL,
    NumOfParDepPlots = 10L,

    # Grid tuning setup
    PassInGrid = NULL,
    GridTune = FALSE,
    BaselineComparison = 'default',
    MaxModelsInGrid = 1L,
    MaxRunsWithoutNewWinner = 20L,
    MaxRunMinutes = 60L*60L,
    MetricPeriods = 25L,

    # Bandit grid args
    Langevin = FALSE,
    DiffusionTemperature = 10000,
    Trees = list('classifier' = 50, 'regression' = 50),
    Depth = list('classifier' = 4, 'regression' = 4),
    RandomStrength = list('classifier' = 1, 'regression' = 1),
    BorderCount = list('classifier' = 32, 'regression' = 32),
    LearningRate = list('classifier' = 0.01, 'regression' = 0.01),
    L2_Leaf_Reg = list('classifier' = 3.0, 'regression' = 1.0),
    RSM = list('classifier' = 0.80, 'regression' = 0.80),
    BootStrapType = list('classifier' = 'Bayesian', 'regression' = 'Bayesian'),
  GrowPolicy = list('classifier' = 'SymmetricTree', 'regression' = 'SymmetricTree'))}, error = function(x) NUL

  # Outcome
  if(!is.null(TestModel)) CatBoost_QA[run, Success := "Success"]
  TestModel <- NULL
  gc(); Sys.sleep(5)
  data.table::fwrite(CatBoost_QA, file = file.path(Path, "AutoCatBoostHurdleModel_QA.csv"))

  # Outcome
  if(!is.null(TestModel)) CatBoost_QA[run, Success := "Success"]
  data.table::fwrite(CatBoost_QA, file = file.path(Path, "AutoCatBoostHurdleModel_QA.csv"))

  # Score CatBoost Hurdle Model
  Output <- tryCatch({RemixAutoML::AutoCatBoostHurdleModelScoring(
    TestData = TTrainData,
    Path = Path,
    ModelID = "ModelTest",
    ModelList = TestModel$ModelList,
```

```
      ArgsList = TestModel$ArgsList,
      Threshold = NULL)}, error = function(x) NULL)

    # Outcome
    if(!is.null(Output)) CatBoost_QA[run, ScoreSuccess := "Success"]
    TestModel <- NULL
    Output <- NULL
    gc(); Sys.sleep(5)
   data.table::fwrite(CatBoost_QA, file = file.path(Path, "AutoCatBoostHurdleModel_QA.csv"))
}

## End(Not run)
```

```
AutoCatBoostHurdleModelScoring
                     AutoCatBoostHurdleModelScoring
```

Description

AutoCatBoostHurdleModelScoring can score AutoCatBoostHurdleModel() models

Usage

```
AutoCatBoostHurdleModelScoring(
  TestData = NULL,
  Path = NULL,
  ModelID = NULL,
  ArgsList = NULL,
  ModelList = NULL,
  Threshold = NULL,
  CARMA = FALSE
)
```

Arguments

TestData	scoring data.table
Path	Supply if ArgsList is NULL or ModelList is null.
ModelID	Supply if ArgsList is NULL or ModelList is null. Same as used in model training.
ArgsList	Output from the hurdle model
ModelList	Output from the hurdle model
Threshold	NULL to use raw probabilities to predict. Otherwise, supply a threshold
CARMA	Keep FALSE. Used for CARMA functions internals

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 Hurdle Modeling: `AutoLightGBMHurdleModelScoring()`, `AutoXGBoostHurdleModelScoring`

Examples

```
## Not run:

# Define file path
Path <- getwd()

# 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::AutoCatBoostHurdleModel(

  # 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::AutoCatBoostHurdleModelScoring(
  TestData = data,
  Path = Path,
  ModelID = "ModelTest",
  ModelList = NULL,
  ArgsList = NULL,
  Threshold = NULL)

## End(Not run)
```

AutoCatBoostMultiClass

AutoCatBoostMultiClass

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(
 OutputSelection = c("Importances", "EvalPlots", "EvalMetrics", "Score_TrainData"),
  data = NULL,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  PrimaryDateColumn = NULL,
  WeightsColumnName = NULL,
```

```
    IDcols = NULL,
    TrainOnFull = FALSE,
    task_type = "GPU",
    NumGPUs = 1,
    DebugMode = FALSE,
    ReturnModelObjects = TRUE,
    SaveModelObjects = FALSE,
    ModelID = "FirstModel",
    model_path = NULL,
    metadata_path = NULL,
    ClassWeights = NULL,
    NumOfParDepPlots = 3,
    eval_metric = "MultiClassOneVsAll",
    loss_function = "MultiClassOneVsAll",
    grid_eval_metric = "Accuracy",
    BaselineComparison = "default",
    MetricPeriods = 10L,
    PassInGrid = NULL,
    GridTune = FALSE,
    MaxModelsInGrid = 30L,
    MaxRunsWithoutNewWinner = 20L,
    MaxRunMinutes = 24L * 60L,
    Trees = 50L,
    Depth = 6,
    LearningRate = NULL,
    L2_Leaf_Reg = NULL,
    RandomStrength = 1,
    BorderCount = 128,
    RSM = NULL,
    BootStrapType = NULL,
    GrowPolicy = NULL,
    langevin = FALSE,
    diffusion_temperature = 10000,
    model_size_reg = 0.5,
    feature_border_type = "GreedyLogSum",
    sampling_unit = "Object",
    subsample = NULL,
    score_function = "Cosine",
    min_data_in_leaf = 1
)
```

Arguments

OutputSelection

You can select what type of output you want returned. Choose from c('Importances', 'EvalPlots', 'EvalMetrics', 'PDFs', 'Score_TrainData')

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

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

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

this data set.

TargetColumnName

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

FeatureColNames

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

PrimaryDateColumn

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

WeightsColumnName

Supply a column name for your weights column. Leave NULL otherwise

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

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

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

NumGPUs   Set to 1, 2, 3, etc.

DebugMode   TRUE to print out steps taken

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

ModelID   A character string to name your model and output

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.

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.

NumOfParDepPlots

Number of partial dependence plots to create for each target level

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'

grid_eval_metric

For evaluating models within grid tuning. Choices include, 'accuracy', 'microauc', 'logloss'

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

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

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')

langevin   TRUE or FALSE. Enable stochastic gradient langevin boosting

diffusion_temperature
> Default is 10000 and is only used when langevin is set to TRUE

model_size_reg   Defaults to 0.5. Set to 0 to allow for bigger models. This is for models with high cardinality categorical features. Valuues greater than 0 will shrink the model and quality will decline but models won't be huge.

feature_border_type
> Defaults to 'GreedyLogSum'. Other options include: Median, Uniform, UniformAndQuantiles, MaxLogSum, MinEntropy

sampling_unit   Default is Group. Other option is Object. if GPU is selected, this will be turned off unless the loss_function is YetiRankPairWise

subsample   Default is NULL. Catboost will turn this into 0.66 for BootStrapTypes Poisson and Bernoulli. 0.80 for MVS. Doesn't apply to others.

score_function Default is Cosine. CPU options are Cosine and L2. GPU options are Cosine,
                L2, NewtonL2, and NewtomCosine (not available for Lossguide)

min_data_in_leaf

                Default is 1. Cannot be used with SymmetricTree is GrowPolicy

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',
    NumGPUs = 1,
    TrainOnFull = FALSE,
    DebugMode = FALSE,

    # Metadata args
    OutputSelection = c('Importances', 'EvalPlots', 'EvalMetrics', 'Score_TrainData'),
    ModelID = 'Test_Model_1',
    model_path = normalizePath('./'),
    metadata_path = normalizePath('./'),
    SaveModelObjects = FALSE,
    ReturnModelObjects = TRUE,

    # Data args
    data = data,
    ValidationData = NULL,
    TestData = NULL,
    TargetColumnName = 'Adrian',
    FeatureColNames = names(data)[!names(data) %in%
```

```
        c('IDcol_1', 'IDcol_2','Adrian')],
    PrimaryDateColumn = NULL,
    WeightsColumnName = NULL,
    ClassWeights = c(1L,1L,1L,1L,1L),
    IDcols = c('IDcol_1','IDcol_2'),

    # Model evaluation
    eval_metric = 'MCC',
    loss_function = 'MultiClassOneVsAll',
    grid_eval_metric = 'Accuracy',
    MetricPeriods = 10L,
    NumOfParDepPlots = 3,

    # Grid tuning args
    PassInGrid = NULL,
    GridTune = FALSE,
    MaxModelsInGrid = 30L,
    MaxRunsWithoutNewWinner = 20L,
    MaxRunMinutes = 24L*60L,
    BaselineComparison = 'default',

    # ML args
    langevin = FALSE,
    diffusion_temperature = 10000,
    Trees = 100L,
    Depth = 4L,
    LearningRate = NULL,
    L2_Leaf_Reg = NULL,
    RandomStrength = 1,
    BorderCount = 254,
    RSM = NULL,
    BootStrapType = 'Bayesian',
    GrowPolicy = 'SymmetricTree',
    model_size_reg = 0.5,
    feature_border_type = 'GreedyLogSum',
    sampling_unit = 'Object',
    subsample = NULL,
    score_function = 'Cosine',
    min_data_in_leaf = 1)

## End(Not run)
```

AutoCatBoostRegression

AutoCatBoostRegression

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(
 OutputSelection = c("Importances", "EvalPlots", "EvalMetrics", "Score_TrainData"),
 data,
 ValidationData = NULL,
 TestData = NULL,
 TargetColumnName = NULL,
 FeatureColNames = NULL,
 PrimaryDateColumn = NULL,
 WeightsColumnName = NULL,
 IDcols = NULL,
 TransformNumericColumns = NULL,
 Methods = c("BoxCox", "Asinh", "Log", "LogPlus1", "Sqrt", "Asin", "Logit"),
 TrainOnFull = FALSE,
 task_type = "GPU",
 NumGPUs = 1,
 DebugMode = FALSE,
 ReturnModelObjects = TRUE,
 SaveModelObjects = FALSE,
 ModelID = "FirstModel",
 model_path = NULL,
 metadata_path = NULL,
 SaveInfoToPDF = FALSE,
 eval_metric = "RMSE",
 eval_metric_value = 1.5,
 loss_function = "RMSE",
 loss_function_value = 1.5,
 grid_eval_metric = "r2",
 NumOfParDepPlots = 0L,
 PassInGrid = NULL,
 GridTune = FALSE,
 MaxModelsInGrid = 30L,
 MaxRunsWithoutNewWinner = 20L,
 MaxRunMinutes = 24L * 60L,
 BaselineComparison = "default",
 MetricPeriods = 10L,
 Trees = 500L,
 Depth = 9,
 L2_Leaf_Reg = 3,
 RandomStrength = 1,
 BorderCount = 254,
 LearningRate = NULL,
 RSM = 1,
 BootStrapType = NULL,
 GrowPolicy = "SymmetricTree",
 langevin = FALSE,
 diffusion_temperature = 10000,
 model_size_reg = 0.5,
 feature_border_type = "GreedyLogSum",
 sampling_unit = "Object",
 subsample = NULL,
 score_function = "Cosine",
```

```
    min_data_in_leaf = 1
)
```

Arguments

OutputSelection

You can select what type of output you want returned. Choose from c('Importances',
'EvalPlots', 'EvalMetrics', 'PDFs', 'Score_TrainData')

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

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

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

TargetColumnName

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

FeatureColNames

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

PrimaryDateColumn

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

WeightsColumnName

Supply a column name for your weights column. Leave NULL otherwise

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 'YeoJohnson', 'BoxCox', 'Asinh', 'Log', 'LogPlus1', 'Sqrt', 'Asin',
or 'Logit'. If more than one is selected, the one with the best normalization pearson statistic will be used. Identity is automatically selected and compared.

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

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

NumGPUs   Set to 1, 2, 3, etc.

DebugMode   Set to TRUE to get a printout of which step the function is on. FALSE, otherwise

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

ModelID   A character string to name your model and output

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

eval_metric	Select from 'RMSE', 'MAE', 'MAPE', 'R2', 'Poisson', 'MedianAbsoluteError', 'SMAPE', 'MSLE', 'NumErrors', 'FairLoss', 'Tweedie', 'Huber', 'LogLinQuantile', 'Quantile', 'Lq', 'Expectile', 'MultiRMSE'
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', 'MultiRMSE'
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
grid_eval_metric	
	Choose from 'mae', 'mape', 'rmse', 'r2'. Case sensitive
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)
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
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
Trees	Standard + Grid Tuning. Bandit grid partitioned. The maximum number of trees you want in your models
Depth	Standard + Grid Tuning. 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	Standard + Grid Tuning. 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	Standard + Grid Tuning. A multiplier of randomness added to split evaluations. Default value is 1 which adds no randomness.
BorderCount	Standard + Grid Tuning. Number of splits for numerical features. Catboost defaults to 254 for CPU and 128 for GPU

LearningRate	Standard + Grid Tuning. Default varies if RMSE, MultiClass, or Logloss is utilized. Otherwise default is 0.03. 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. Standard + Grid Tuning. If GPU is set, this is turned off. 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	Standard + Grid Tuning. NULL value to default to catboost default (Bayesian for GPU and MVS for CPU). 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	Standard + Grid Tuning. Catboost default of SymmetricTree. Random testing. Default 'SymmetricTree', 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')
langevin	Set to TRUE to enable
diffusion_temperature	Defaults to 10000
model_size_reg	Defaults to 0.5. Set to 0 to allow for bigger models. This is for models with high cardinality categorical features. Valuues greater than 0 will shrink the model and quality will decline but models won't be huge.
feature_border_type	Defaults to 'GreedyLogSum'. Other options include: Median, Uniform, UniformAndQuantiles, MaxLogSum, MinEntropy
sampling_unit	Default is Group. Other option is Object. if GPU is selected, this will be turned off unless the loss_function is YetiRankPairWise
subsample	Default is NULL. Catboost will turn this into 0.66 for BootStrapTypes Poisson and Bernoulli. 0.80 for MVS. Doesn't apply to others.
score_function	Default is Cosine. CPU options are Cosine and L2. GPU options are Cosine, L2, NewtonL2, and NewtomCosine (not available for Lossguide)
min_data_in_leaf	Default is 1. Cannot be used with SymmetricTree is GrowPolicy

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()`, `AutoLightGBMRegression()`, `AutoNLS()`, `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
  TrainOnFull = FALSE,
  task_type = 'GPU',
  NumGPUs = 1,
  DebugMode = FALSE,

  # Metadata args
  OutputSelection = c('Importances', 'EvalPlots', 'EvalMetrics', 'Score_TrainData'),
  ModelID = 'Test_Model_1',
  model_path = normalizePath('./'),
  metadata_path = normalizePath('./'),
  SaveModelObjects = FALSE,
  SaveInfoToPDF = FALSE,
  ReturnModelObjects = TRUE,

  # Data args
  data = data,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = 'Adrian',
  FeatureColNames = names(data)[!names(data) %in%
    c('IDcol_1', 'IDcol_2','Adrian')],
  PrimaryDateColumn = NULL,
  WeightsColumnName = NULL,
  IDcols = c('IDcol_1','IDcol_2'),
  TransformNumericColumns = 'Adrian',
  Methods = c('BoxCox', 'Asinh', 'Asin', 'Log',
    'LogPlus1', 'Sqrt', 'Logit'),

  # Model evaluation
  eval_metric = 'RMSE',
  eval_metric_value = 1.5,
  loss_function = 'RMSE',
  loss_function_value = 1.5,
  MetricPeriods = 10L,
```

```
  NumOfParDepPlots = ncol(data)-1L-2L,

  # Grid tuning args
  PassInGrid = NULL,
  GridTune = FALSE,
  MaxModelsInGrid = 30L,
  MaxRunsWithoutNewWinner = 20L,
  MaxRunMinutes = 60*60,
  BaselineComparison = 'default',

  # ML args
  langevin = FALSE,
  diffusion_temperature = 10000,
  Trees = 1000,
  Depth = 9,
  L2_Leaf_Reg = NULL,
  RandomStrength = 1,
  BorderCount = 128,
  LearningRate = NULL,
  RSM = 1,
  BootStrapType = NULL,
  GrowPolicy = 'SymmetricTree',
  model_size_reg = 0.5,
  feature_border_type = 'GreedyLogSum',
  sampling_unit = 'Object',
  subsample = NULL,
  score_function = 'Cosine',
  min_data_in_leaf = 1)

## 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 = FALSE,
    MDP_CharToFactor = FALSE,
    MDP_RemoveDates = FALSE,
    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(), AutoCatBoostClassifier() 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(), AutoLightGBMScoring(), AutoXGBoostScoring()

Examples

```
## Not run:

# CatBoost Regression Example

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

```
  MultiClass = FALSE)

# Copy data
data1 <- data.table::copy(data)

# Run function
TestModel <- RemixAutoML::AutoCatBoostRegression(

  # GPU or CPU and the number of available GPUs
  TrainOnFull = FALSE,
  task_type = 'CPU',
  NumGPUs = 1,
  DebugMode = FALSE,

  # Metadata args
  OutputSelection = c('Importances','EvalPlots','EvalMetrics','Score_TrainData'),
  ModelID = 'Test_Model_1',
  model_path = getwd(),
  metadata_path = getwd(),
  SaveModelObjects = FALSE,
  SaveInfoToPDF = FALSE,
  ReturnModelObjects = TRUE,

  # Data args
  data = data1,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = 'Adrian',
  FeatureColNames = names(data1)[!names(data1) %in% c('IDcol_1', 'IDcol_2','Adrian')],
  PrimaryDateColumn = NULL,
  WeightsColumnName = NULL,
  IDcols = c('IDcol_1','IDcol_2'),
  TransformNumericColumns = 'Adrian',
  Methods = c('Asinh','Asin','Log','LogPlus1','Sqrt','Logit'),

  # Model evaluation
  eval_metric = 'RMSE',
  eval_metric_value = 1.5,
  loss_function = 'RMSE',
  loss_function_value = 1.5,
  MetricPeriods = 10L,
  NumOfParDepPlots = ncol(data1)-1L-2L,

  # Grid tuning args
  PassInGrid = NULL,
  GridTune = FALSE,
  MaxModelsInGrid = 30L,
  MaxRunsWithoutNewWinner = 20L,
  MaxRunMinutes = 60*60,
  BaselineComparison = 'default',

  # ML args
  langevin = FALSE,
  diffusion_temperature = 10000,
  Trees = 1000,
  Depth = 9,
  L2_Leaf_Reg = NULL,
```

```
    RandomStrength = 1,
    BorderCount = 128,
    LearningRate = NULL,
    RSM = 1,
    BootStrapType = NULL,
    GrowPolicy = 'SymmetricTree',
    model_size_reg = 0.5,
    feature_border_type = 'GreedyLogSum',
    sampling_unit = 'Object',
    subsample = NULL,
    score_function = 'Cosine',
    min_data_in_leaf = 1)

# Trained Model Object
TestModel$Model

# Train Data (includes validation data) and Test Data with predictions and shap values
TestModel$TrainData
TestModel$TestData

# Calibration Plots
TestModel$PlotList$Train_EvaluationPlot
TestModel$PlotList$Test_EvaluationPlot

# Calibration Box Plots
TestModel$PlotList$Train_EvaluationBoxPlot
TestModel$PlotList$Test_EvaluationBoxPlot

# Residual Analysis Plots
TestModel$PlotList$Train_ResidualsHistogram
TestModel$PlotList$Test_ResidualsHistogram

# Preds vs Actuals Scatterplots
TestModel$PlotList$Train_ScatterPlot
TestModel$PlotList$Test_ScatterPlot

# Preds vs Actuals Copula Plot
TestModel$PlotList$Train_CopulaPlot
TestModel$PlotList$Test_CopulaPlot

# Variable Importance Plots
TestModel$PlotList$Train_VariableImportance
TestModel$PlotList$Validation_VariableImportance
TestModel$PlotList$Test_VariableImportance

# Evaluation Metrics
TestModel$EvaluationMetrics$TrainData
TestModel$EvaluationMetrics$TestData

# Variable Importance Tables
TestModel$VariableImportance$Train_Importance
TestModel$VariableImportance$Validation_Importance
TestModel$VariableImportance$Test_Importance

# Interaction Importance
TestModel$InteractionImportance$Train_Interaction
TestModel$InteractionImportance$Validation_Interaction
```

```
TestModel$InteractionImportance$Test_Interaction

# Meta Data
TestModel$ColNames
TestModel$TransformationResults
TestModel$GridList

# Score data
Preds <- RemixAutoML::AutoCatBoostScoring(
  TargetType = 'regression',
  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,
  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)
```

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,
  NonNegativePred = FALSE,
```

```
RoundPreds = FALSE,
TrainOnFull = FALSE,
TargetColumnName = "Target",
DateColumnName = "DateTime",
HierarchGroups = NULL,
GroupVariables = NULL,
TimeWeights = 1,
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"),
HolidayLookback = NULL,
HolidayLags = 1L,
HolidayMovingAverages = 1L:2L,
TimeTrendVariable = FALSE,
ZeroPadSeries = NULL,
DataTruncate = FALSE,
SplitRatios = c(0.7, 0.2, 0.1),
TaskType = "GPU",
NumGPU = 1,
PartitionType = "timeseries",
Timer = TRUE,
DebugMode = FALSE,
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 = NULL,
    LearningRate = NULL,
    RandomStrength = 1,
    BorderCount = 254,
    Depth = 6,
    RSM = 1,
    BootStrapType = "Bayesian",
    GrowPolicy = "SymmetricTree",
    ModelSizeReg = 0.5,
    FeatureBorderType = "GreedyLogSum",
    SamplingUnit = "Group",
    SubSample = NULL,
    ScoreFunction = "Cosine",
    MinDataInLeaf = 1
)
```

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 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.
`TimeWeights`	NULL or a value.
`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'
HolidayLookback	
	Number of days in range to compute number of holidays from a given date in the data. If NULL, the number of days are computed for you.
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.
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
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	Enables the Stochastic Gradient Langevin Boosting mode. If TRUE and Task-Type == '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
LearningRate	Defaults to NULL. Catboost will dynamically define this if L2_Leaf_Reg is NULL and RMSE is chosen (otherwise catboost will default it to 0.03). Then you can pull it out of the model object and pass it back in should you wish.
RandomStrength	Default is 1
BorderCount	Default is 254
Depth	Depth of catboost model
RSM	CPU only. If TaskType is GPU then RSM will not be used
BootStrapType	If NULL, then if TaskType is GPU then Bayesian will be used. If CPU then MVS will be used. If MVS is selected when TaskType is GPU, then BootStrap-Type will be switched to Bayesian
GrowPolicy	Default is SymmetricTree. Others include Lossguide and Depthwise
ModelSizeReg	Defaults to 0.5. Set to 0 to allow for bigger models. This is for models with high cardinality categorical features. Valuues greater than 0 will shrink the model and quality will decline but models won't be huge.
FeatureBorderType	Defaults to 'GreedyLogSum'. Other options include: Median, Uniform, Uniform-mAndQuantiles, MaxLogSum, MinEntropy
SamplingUnit	Default is Group. Other option is Object. if GPU is selected, this will be turned off unless the loss_function is YetiRankPairWise
SubSample	Can use if BootStrapType is neither Bayesian nor No. Pass NULL to use Cat-boost default. Used for bagging.
ScoreFunction	Default is Cosine. CPU options are Cosine and L2. GPU options are Cosine, L2, NewtonL2, and NewtomCosine (not available for Lossguide)
MinDataInLeaf	Defaults to 1. Used if GrowPolicy is not SymmetricTree

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(), AutoLightGBMCARMA(), AutoLightGBMHurdleCARMA(), AutoXGBoostCARMA(), AutoXGBoostHurdleCARMA()

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,
  TargetColumnName = c('Weekly_Sales','Weekly_Profit'),
  DateColumnName = 'Date',
  HierarchGroups = NULL,
  GroupVariables = c('Store','Dept'),
  TimeWeights = 1,
```

```
          TimeUnit = 'weeks',
          TimeGroups = c('weeks','months'),

          # Production args
          TaskType = 'GPU',
          NumGPU = 1,
          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'),
          HolidayLookback = NULL,
          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,

          # Eval args
          NumOfParDepPlots = 100L,
          EvalMetric = 'MultiRMSE',
          EvalMetricValue = 1.5,
          LossFunction = 'MultiRMSE',
          LossFunctionValue = 1.5,

          # Grid args
```

```
      GridTune = FALSE,
      PassInGrid = NULL,
      ModelCount = 5,
      MaxRunsWithoutNewWinner = 50,
      MaxRunMinutes = 60*60,

      # ML Args
      NTrees = 1000,
      Depth = 6,
      LearningRate = NULL,
      L2_Leaf_Reg = NULL,
      RandomStrength = 1,
      BorderCount = 254,
      RSM = 1,
      BootStrapType = 'Bayesian',
      GrowPolicy = 'SymmetricTree',
      Langevin = FALSE,
      DiffusionTemperature = 10000,
      ModelSizeReg = 0.5,
      FeatureBorderType = 'GreedyLogSum',
      SamplingUnit = 'Group',
      SubSample = NULL,
      ScoreFunction = 'Cosine',
      MinDataInLeaf = 1)

   ## End(Not run)
```

AutoClustering   *AutoClustering*

Description

AutoClustering adds a column to your original data with a cluster number identifier. You can run request an autoencoder to be built to reduce the dimensionality of your data before running the clusering algo.

Usage

```
AutoClustering(
  data,
  FeatureColumns = NULL,
  ModelID = "TestModel",
  SavePath = NULL,
  NThreads = 8,
  MaxMemory = "28G",
  MaxClusters = 50,
  ClusterMetric = "totss",
  RunDimReduction = TRUE,
  ShrinkRate = (sqrt(5) - 1)/2,
  Epochs = 5L,
  L2_Reg = 0.1,
  ElasticAveraging = TRUE,
  ElasticAveragingMovingRate = 0.9,
```

```
    ElasticAveragingRegularization = 0.001
  )
```

Arguments

data	is the source time series data.table
FeatureColumns	Independent variables
ModelID	For naming the files to save
SavePath	Directory path for saving models
NThreads	set based on number of threads your machine has available
MaxMemory	set based on the amount of memory your machine has available
MaxClusters	number of factors to test out in k-means to find the optimal number
ClusterMetric	pick the metric to identify top model in grid tune c('totss','betweenss','withinss')
RunDimReduction	
	If TRUE, an autoencoder will be built to reduce the feature space. Otherwise, all features in FeatureColumns will be used for clustering
ShrinkRate	Node shrink rate for H2OAutoencoder. See that function for details.
Epochs	For the autoencoder
L2_Reg	For the autoencoder
ElasticAveraging	
	For the autoencoder
ElasticAveragingMovingRate	
	For the autoencoder
ElasticAveragingRegularization	
	For the autoencoder

Value

Original data.table with added column with cluster number identifier

Author(s)

Adrian Antico

See Also

Other Unsupervised Learning: `AutoClusteringScoring`(), `GenTSAnomVars`(), `H2OIsolationForestScoring`(), `H2OIsolationForest`(), `ResidualOutliers`()

Examples

```
## Not run:
#########################
# Training Setup
#########################

# Create fake data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000,
  ID = 2,
```

```
    ZIP = 0,
    AddDate = TRUE,
    Classification = FALSE,
    MultiClass = FALSE)

  # Run function
  data <- RemixAutoML::AutoClustering(
    data,
    FeatureColumns = names(data)[2:(ncol(data)-1)],
    ModelID = 'TestModel',
    SavePath = getwd(),
    NThreads = 8,
    MaxMemory = '28G',
    MaxClusters = 50,
    ClusterMetric = 'totss',
    RunDimReduction = TRUE,
    ShrinkRate = (sqrt(5) - 1) / 2,
    Epochs = 5L,
    L2_Reg = 0.10,
    ElasticAveraging = TRUE,
    ElasticAveragingMovingRate = 0.90,
    ElasticAveragingRegularization = 0.001)

  #########################
  # Scoring Setup
  #########################

  Sys.sleep(10)

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

  # Run function
  data <- RemixAutoML::AutoClusteringScoring(
    data,
    FeatureColumns = names(data)[2:(ncol(data)-1)],
    ModelID = 'TestModel',
    SavePath = getwd(),
    NThreads = 8,
    MaxMemory = '28G',
    DimReduction = TRUE)

  ## End(Not run)
```

AutoClusteringScoring   *AutoClusteringScoring*

Description

AutoClusteringScoring adds a column to your original data with a cluster number identifier. You can run request an autoencoder to be built to reduce the dimensionality of your data before running the clusering algo.

Usage

```
AutoClusteringScoring(
  data,
  FeatureColumns = NULL,
  ModelID = "TestModel",
  SavePath = NULL,
  NThreads = 8,
  MaxMemory = "28G",
  DimReduction = TRUE
)
```

Arguments

data	is the source time series data.table
FeatureColumns	Independent variables
ModelID	This is returned from the training run in the output list with element named 'model_name'. It's not identical to the ModelID used in training due to the grid tuning.
SavePath	Directory path for saving models
NThreads	set based on number of threads your machine has available
MaxMemory	set based on the amount of memory your machine has available
DimReduction	Set to TRUE if you set RunDimReduction in the training version of this function

Value

Original data.table with added column with cluster number identifier

Author(s)

Adrian Antico

See Also

Other Unsupervised Learning: AutoClustering(), GenTSAnomVars(), H2OIsolationForestScoring(), H2OIsolationForest(), ResidualOutliers()

Examples

```
## Not run:
########################
# Training Setup
########################

# Create fake data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
```

```
    N = 1000,
    ID = 2,
    ZIP = 0,
    AddDate = TRUE,
    Classification = FALSE,
    MultiClass = FALSE)

  # Run function
  data <- RemixAutoML::AutoClustering(
    data,
    FeatureColumns = names(data)[2:(ncol(data)-1)],
    ModelID = 'TestModel',
    SavePath = getwd(),
    NThreads = 8,
    MaxMemory = '28G',
    MaxClusters = 50,
    ClusterMetric = 'totss',
    RunDimReduction = TRUE,
    ShrinkRate = (sqrt(5) - 1) / 2,
    Epochs = 5L,
    L2_Reg = 0.10,
    ElasticAveraging = TRUE,
    ElasticAveragingMovingRate = 0.90,
    ElasticAveragingRegularization = 0.001)

  #########################
  # Scoring Setup
  #########################

  Sys.sleep(10)

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

  # Run function
  data <- RemixAutoML::AutoClusteringScoring(
    data,
    FeatureColumns = names(data)[2:(ncol(data)-1)],
    ModelID = 'TestModel',
    SavePath = getwd(),
    NThreads = 8,
    MaxMemory = '28G',
    DimReduction = TRUE)

  ## 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 Database: `SQL_ClearTable()`, `SQL_DropTable()`, `SQL_Query_Push()`, `SQL_Query()`, `SQL_SaveTable()`, `SQL_Server_DBConnection()`

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,
  TimeColumnName = NULL
)
```

Arguments

`data`	Source data to do your partitioning on
`NumDataSets`	The number of total data sets you want built
`Ratios`	A vector of values for how much data each data set should get in each split. E.g. c(0.70, 0.20, 0.10)
`PartitionType`	Set to either "random", "timeseries", or "time". With "random", your data will be paritioned randomly (with stratified sampling if column names are supplied). With "timeseries", you can partition by time with a stratify option (so long as you have an equal number of records for each strata). With "time" you will have data sets generated so that the training data contains the earliest records in time, validation data the second earliest, test data the third earliest, etc.
`StratifyColumnNames`	
	Supply column names of categorical features to use in a stratified sampling procedure for partitioning the data. Partition type must be "random" to use this option
`TimeColumnName`	Supply a date column name or a name of a column with an ID for sorting by time such that the smallest number is the earliest in time.

Value

Returns a list of data.tables

Author(s)

Adrian Antico

See Also

Other Feature Engineering: `AutoDiffLagN()`, `AutoHierarchicalFourier()`, `AutoInteraction()`, `AutoLagRollMode()`, `AutoLagRollStatsScoring()`, `AutoLagRollStats()`, `AutoTransformationCreate()`, `AutoTransformationScore()`, `AutoWord2VecModeler()`, `AutoWord2VecScoring()`, `CategoricalEncoding()`, `CreateCalendarVariables()`, `CreateHolidayVariables()`, `DummifyDT()`, `H2OAutoencoderScoring()`, `H2OAutoencoder()`, `ModelDataPrep()`, `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,
```

```
    TimeColumnName = NULL)

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

AutoDiffLagN   *AutoDiffLagN*

Description

AutoDiffLagN create differences for selected numerical columns

Usage

```
AutoDiffLagN(
  data,
  DateVariable = NULL,
  GroupVariables = NULL,
  DiffVariables = NULL,
  DiffDateVariables = NULL,
  DiffGroupVariables = NULL,
  NLag1 = 0L,
  NLag2 = 1L,
  Sort = FALSE,
  RemoveNA = TRUE
)
```

Arguments

`data`	Source data
`DateVariable`	Date column used for sorting
`GroupVariables`	Difference data by group
`DiffVariables`	Column names of numeric columns to difference
`DiffDateVariables`	
	Columns names for date variables to difference. Output is a numeric value representing the difference in days.
`DiffGroupVariables`	
	Column names for categorical variables to difference. If no change then the output is 'No_Change' else 'New=NEWVAL Old=OLDVAL' where NEWVAL and OLDVAL are placeholders for the actual values
`NLag1`	If the diff calc, we have column 1 - column 2. NLag1 is in reference to column 1. If you want to take the current value minus the previous weeks value, supply a zero. If you want to create a lag2 - lag4 NLag1 gets a 2.
`NLag2`	If the diff calc, we have column 1 - column 2. NLag2 is in reference to column 2. If you want to take the current value minus the previous weeks value, supply a 1. If you want to create a lag2 - lag4 NLag1 gets a 4.
`Sort`	TRUE to sort your data inside the function
`RemoveNA`	Set to TRUE to remove rows with NA generated by the lag operation

Author(s)

Adrian Antico

See Also

Other Feature Engineering: AutoDataPartition(), AutoHierarchicalFourier(), AutoInteraction(),
AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(),
AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring(), CategoricalEncoding(),
CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(),
H2OAutoencoder(), ModelDataPrep(), TimeSeriesFill()

Examples

```
## Not run:

# Create fake data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.70,
  N = 50000,
  ID = 2L,
  FactorCount = 3L,
  AddDate = TRUE,
  ZIP = 0L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Store Cols to diff
Cols <- names(data)[which(unlist(data[, lapply(.SD, is.numeric)]))]

# Clean data before running AutoDiffLagN
data <- RemixAutoML::ModelDataPrep(data = data, Impute = FALSE, CharToFactor = FALSE, FactorToChar = TRUE)

# Run function
data <- RemixAutoML::AutoDiffLagN(
  data,
  DateVariable = "DateTime",
  GroupVariables = c("Factor_1", "Factor_2"),
  DiffVariables = Cols,
  DiffDateVariables = NULL,
  DiffGroupVariables = NULL,
  NLag1 = 0L,
  NLag2 = 1L,
  Sort = TRUE,
  RemoveNA = TRUE)

## End(Not run)
```

AutoETS   *AutoETS*

Description

AutoETS 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

```
AutoETS(
  data,
  FilePath = NULL,
  TargetVariableName,
  DateColumnName,
  TimeAggLevel = "week",
  EvaluationMetric = "MAE",
  NumHoldOutPeriods = 5L,
  NumFCPeriods = 5L,
  TrainWeighting = 0.5,
  MaxConsecutiveFails = 12L,
  MaxNumberModels = 100L,
  MaxRunTimeMinutes = 10L,
  NumberCores = max(1L, min(4L, parallel::detectCores() - 2L))
)
```

Arguments

`data`	Source data.table
`FilePath`	NULL to return nothing. Provide a file path to save the model and xregs if available
`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
`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   Default max(1L, min(4L, parallel::detectCores()-2L))

Author(s)

Adrian Antico

See Also

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

Examples

```
## Not run:
# Create fake data
data <- RemixAutoML::FakeDataGenerator(TimeSeries = TRUE, TimeSeriesTimeAgg = "days")

# Build model
Output <- RemixAutoML::AutoETS(
  data,
  FilePath = NULL,
  TargetVariableName = "Weekly_Sales",
  DateColumnName = "Date",
  TimeAggLevel = "weeks",
  EvaluationMetric = "MAE",
  NumHoldOutPeriods = 5L,
  NumFCPeriods = 5L,
  TrainWeighting = 0.50,
  MaxConsecutiveFails = 12L,
  MaxNumberModels = 100L,
  MaxRunTimeMinutes = 10L,
  NumberCores = max(1L, min(4L, parallel::detectCores()-2L)))

# Output
Output$ForecastPlot
Output$Forecast
Output$PerformanceGrid

## End(Not run)
```

AutoH2OCARMA   *AutoH2OCARMA*

Description

AutoH2OCARMA 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

```
AutoH2OCARMA(
  AlgoType = "drf",
  ExcludeAlgos = "XGBoost",
  data,
  TrainOnFull = FALSE,
  TargetColumnName = "Target",
  PDFOutputPath = NULL,
  SaveDataPath = NULL,
  TimeWeights = NULL,
  NonNegativePred = FALSE,
  RoundPreds = FALSE,
  DateColumnName = "DateTime",
  GroupVariables = NULL,
  HierarchGroups = NULL,
  TimeUnit = "week",
  TimeGroups = c("weeks", "months"),
  FC_Periods = 30,
  PartitionType = "timeseries",
  MaxMem = {      gc()
   paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo",
     intern = TRUE))/1e+06)), "G") },
  NThreads = max(1, parallel::detectCores() - 2),
  Timer = TRUE,
  DebugMode = FALSE,
  TargetTransformation = FALSE,
  Methods = c("YeoJohnson", "BoxCox", "Asinh", "Log", "LogPlus1", "Sqrt", "Asin",
     "Logit"),
  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",
     "wom", "isoweek", "month", "quarter", "year"),
  HolidayVariable = c("USPublicHolidays", "EasterGroup", "ChristmasGroup",
     "OtherEcclesticalFeasts"),
  HolidayLookback = NULL,
```

```
    HolidayLags = 1,
    HolidayMovingAverages = 1:2,
    TimeTrendVariable = FALSE,
    DataTruncate = FALSE,
    ZeroPadSeries = NULL,
    SplitRatios = c(0.7, 0.2, 0.1),
    EvalMetric = "rmse",
    NumOfParDepPlots = 0L,
    GridTune = FALSE,
    ModelCount = 1,
    NTrees = 1000,
    LearnRate = 0.1,
    LearnRateAnnealing = 1,
    GridStrategy = "Cartesian",
    MaxRunTimeSecs = 60 * 60 * 24,
    StoppingRounds = 10,
    MaxDepth = 20,
    SampleRate = 0.632,
    MTries = -1,
    ColSampleRate = 1,
    ColSampleRatePerTree = 1,
    ColSampleRatePerTreeLevel = 1,
    MinRows = 1,
    NBins = 20,
    NBinsCats = 1024,
    NBinsTopLevel = 1024,
    CategoricalEncoding = "AUTO",
    HistogramType = "AUTO",
    Distribution = "gaussian",
    Link = "identity",
    RandomDistribution = NULL,
    RandomLink = NULL,
    Solver = "AUTO",
    Alpha = NULL,
    Lambda = NULL,
    LambdaSearch = FALSE,
    NLambdas = -1,
    Standardize = TRUE,
    RemoveCollinearColumns = FALSE,
    InterceptInclude = TRUE,
    NonNegativeCoefficients = FALSE,
    RandomColNumbers = NULL,
    InteractionColNumbers = NULL
)
```

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
`TrainOnFull`	Set to TRUE to train on full data
`TargetColumnName`	
	List the column name of your target variables column. E.g. "Target"
`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
`TimeWeights`	1 or a value between zero and 1. Data will be weighted less and less the more historic it gets, by group
`NonNegativePred`	
	TRUE or FALSE
`RoundPreds`	Rounding predictions to an integer value. TRUE or FALSE. Defaults to FALSE
`DateColumnName`	List the column name of your date column. E.g. "DateTime"
`GroupVariables`	Defaults to NULL. Use NULL when you have a single series. Add in Group-Variables when you have a series for every level of a group or multiple groups.
`HierarchGroups`	Vector of hierachy categorical columns.
`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.
`FC_Periods`	Set the number of periods you want to have forecasts for. E.g. 52 for weekly data to forecast a year ahead
`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
`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 "YeoJohnson", "BoxCox", "Asinh", "Log", "LogPlus1", "Sqrt", "Asin", or "Logit". If more than one is selected, the one with the best normalization pearson statistic will be used. Identity is automatically selected and compared.
`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) or list("day" = c(1:10), "weeks" = c(1:4))
`MA_Periods`	Select the periods for all moving average variables you want to create. E.g. c(1:5,52) or list("day" = c(2:10), "weeks" = c(2:4))
`SD_Periods`	Select the periods for all moving standard deviation variables you want to create. E.g. c(1:5,52) or list("day" = c(2:10), "weeks" = c(2:4))

Skew_Periods	Select the periods for all moving skewness variables you want to create. E.g. c(1:5,52) or list("day" = c(2:10), "weeks" = c(2:4))
Kurt_Periods	Select the periods for all moving kurtosis variables you want to create. E.g. c(1:5,52) or list("day" = c(2:10), "weeks" = c(2:4))
Quantile_Periods	
	Select the periods for all moving quantiles variables you want to create. E.g. c(1:5,52) or list("day" = c(2:10), "weeks" = c(2:4))
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"
HolidayLookback	
	Number of days in range to compute number of holidays from a given date in the data. If NULL, the number of days are computed for you.
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	NULL to do nothing. Otherwise, set to "maxmax", "minmax", "maxmin", "min-min". See [TimeSeriesFill](#) for explanations of each type
SplitRatios	E.g c(0.7,0.2,0.1) for train, validation, and test sets
EvalMetric	Select from "RMSE", "MAE", "MAPE", "Poisson", "Quantile", "LogLinQuan-tile", "Lq", "SMAPE", "R2", "MSLE", "MedianAbsoluteError"
NumOfParDepPlots	
	Set to zeros if you do not want any returned. Can set to a very large value and it will adjust to the max number of features if it's too high
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
LearnRate	Default 0.10, models available include gbm

```
LearnRateAnnealing
```
               Default 1, models available include gbm

`GridStrategy`   Default "Cartesian", models available include

`MaxRunTimeSecs` Default 60*60*24, models available include

`StoppingRounds` Default 10, models available include

`MaxDepth`   Default 20, models available include drf, gbm

`SampleRate`   Default 0.632, models available include drf, gbm

`MTries`   Default 1, models available include drf

`ColSampleRate`   Default 1, model available include gbm

```
ColSampleRatePerTree
```
               Default 1, models available include drf, gbm

```
ColSampleRatePerTreeLevel
```
               Default 1, models available include drf, gbm

`MinRows`   Default 1, models available include drf, gbm

`NBins`   Default 20, models available include drf, gbm

`NBinsCats`   Default 1024, models available include drf, gbm

`NBinsTopLevel`   Default 1024, models available include drf, gbm

`CategoricalEncoding`
               Default "AUTO". Choices include : "AUTO", "Enum", "OneHotInternal", "One-HotExplicit", "Binary", "Eigen", "LabelEncoder", "Sort-ByResponse", "Enum-Limited"

`HistogramType`   Default "AUTO". Select from "AUTO", "UniformAdaptive", "Random", "Quan-tilesGlobal", "RoundRobin"

`Distribution`   Model family

`Link`   Link for model family

```
RandomDistribution
```
               Default NULL

`RandomLink`   Default NULL

`Solver`   Model optimizer

`Alpha`   Default NULL

`Lambda`   Default NULL

`LambdaSearch`   Default FALSE,

`NLambdas`   Default -1

`Standardize`   Default TRUE

```
RemoveCollinearColumns
```
               Default FALSE

```
InterceptInclude
```
               Default TRUE

```
NonNegativeCoefficients
```
               Default FALSE

```
RandomColNumbers
```
               NULL

```
InteractionColNumbers
```
               NULL

Value

See examples

Author(s)

Adrian Antico

See Also

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

Examples

```
## Not run:

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

# Ensure series have no missing dates (also remove series with more than 25% missing values)
data <- RemixAutoML::TimeSeriesFill(
  data,
  DateColumnName = "Date",
  GroupVariables = c("Store","Dept"),
  TimeUnit = "weeks",
  FillType = "maxmax",
  MaxMissingPercent = 0.25,
  SimpleImpute = TRUE)

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

# Remove IsHoliday column
data[, IsHoliday := NULL]

# Create xregs (this is the include the categorical variables instead of utilizing only the interaction of them)
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))]

# 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"),
```

```r
  # Data Wrangling Features
  SplitRatios = c(1 - 10 / 138, 10 / 138),
  PartitionType = "random",

  # Production args
  FC_Periods = 4L,
  TrainOnFull = FALSE,
  MaxMem = {gc();paste0(as.character(floor(max(32, as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo
  NThreads = parallel::detectCores(),
  PDFOutputPath = NULL,
  SaveDataPath = NULL,
  Timer = TRUE,
  DebugMode = TRUE,

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

  # Calendar features
  CalendarVariables = c("week", "wom", "month", "quarter", "year"),
  HolidayVariable = c("USPublicHolidays","EasterGroup",
    "ChristmasGroup","OtherEcclesticalFeasts"),
  HolidayLookback = NULL,
  HolidayLags = 1:7,
  HolidayMovingAverages = 2:7,
  TimeTrendVariable = TRUE,

  # Time series features
  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,

  # Bonus Features
  XREGS = NULL,
  FourierTerms = 2L,
  AnomalyDetection = NULL,
  ZeroPadSeries = NULL,
  DataTruncate = FALSE,

  # ML evaluation args
  EvalMetric = "RMSE",
  NumOfParDepPlots = 0L,

  # ML grid tuning args
  GridTune = FALSE,
  GridStrategy = "Cartesian",
  ModelCount = 5,
  MaxRunTimeSecs = 60*60*24,
```

```
        StoppingRounds = 10,

        # ML Args
        NTrees = 1000L,
        MaxDepth = 20,
        SampleRate = 0.632,
        MTries = -1,
        ColSampleRatePerTree = 1,
        ColSampleRatePerTreeLevel = 1,
        MinRows = 1,
        NBins = 20,
        NBinsCats = 1024,
        NBinsTopLevel = 1024,
        HistogramType = "AUTO",
        CategoricalEncoding = "AUTO",
        RandomColNumbers = NULL,
        InteractionColNumbers = NULL,
        WeightsColumn = NULL,

        # ML args
        Distribution = "gaussian",
        Link = "identity",
        RandomDistribution = NULL,
        RandomLink = NULL,
        Solver = "AUTO",
        Alpha = NULL,
        Lambda = NULL,
        LambdaSearch = FALSE,
        NLambdas = -1,
        Standardize = TRUE,
        RemoveCollinearColumns = FALSE,
        InterceptInclude = TRUE,
        NonNegativeCoefficients = FALSE)

    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*

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(
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
  data = NULL,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  WeightsColumn = NULL,
  MaxMem = {     gc()
 paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo",
   intern = TRUE))/1e+06)), "G") },
  NThreads = max(1L, parallel::detectCores() - 2L),
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  NumOfParDepPlots = 3L,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  SaveInfoToPDF = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = FALSE,
  H2OStartUp = TRUE,
  GridTune = FALSE,
  GridStrategy = "RandomDiscrete",
  MaxRunTimeSecs = 60 * 60 * 24,
  StoppingRounds = 10,
  MaxModelsInGrid = 2,
  DebugMode = FALSE,
  eval_metric = "auc",
  CostMatrixWeights = c(1, 0, 0, 1),
  Trees = 50L,
  MaxDepth = 20L,
  SampleRate = 0.632,
  MTries = -1,
  ColSampleRatePerTree = 1,
  ColSampleRatePerTreeLevel = 1,
  MinRows = 1,
  NBins = 20,
  NBinsCats = 1024,
  NBinsTopLevel = 1024,
```

```
    HistogramType = "AUTO",
    CategoricalEncoding = "AUTO"
)
```

Arguments

`OutputSelection`

You can select what type of output you want returned. Choose from c("EvalMetrics", "PDFs", "Score_TrainData")

`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)

`WeightsColumn`   Column name of a weights column

`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

`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

`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

`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

`H2OStartUp`   Defaults to TRUE which means H2O will be started inside the function

`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.

`GridStrategy`   Default "Cartesian"

```
MaxRunTimeSecs  Default 86400
StoppingRounds  Default 10
MaxModelsInGrid
                Number of models to test from grid options (1080 total possible options)
DebugMode       Set to TRUE to get a printout of each step taken internally
eval_metric     This is the metric used to identify best grid tuned model. Choose from "AUC"
                or "logloss"
CostMatrixWeights
                A vector with 4 elements c(True Positive Cost, False Negative Cost, False Posi-
                tive Cost, True Negative Cost). Default c(1,0,0,1),
Trees           The maximum number of trees you want in your models
MaxDepth        Default 20
SampleRate      Default 0.632
MTries          Default -1 means it will default to number of features divided by 3
ColSampleRatePerTree
                Default 1
ColSampleRatePerTreeLevel
                Default 1
MinRows         Default 1
NBinsCats       Default 1024
NBinsTopLevel   Default 1024
HistogramType   Default "AUTO"
CategoricalEncoding
                Default "AUTO"
```

Value

Saves to file and returned in list: VariableImportance.csv, Model, ValidationData.csv, Evaluation-
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()`, `AutoLightGBMClassifier()`,
`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 args
    MaxMem = {gc();paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo", inter
    NThreads = max(1L, parallel::detectCores() - 2L),
    IfSaveModel = "mojo",
    H2OShutdown = FALSE,
    H2OStartUp = TRUE,

    # Model evaluation args
    eval_metric = "auc",
    NumOfParDepPlots = 3L,
    CostMatrixWeights = c(1,0,0,1),

    # Metadata args
    OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
    model_path = normalizePath("./"),
    metadata_path = NULL,
    ModelID = "FirstModel",
    ReturnModelObjects = TRUE,
    SaveModelObjects = FALSE,
    SaveInfoToPDF = FALSE,
    DebugMode = FALSE,

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

    # Grid Tuning Args
    GridStrategy = "RandomDiscrete",
    GridTune = FALSE,
    MaxModelsInGrid = 10,
    MaxRunTimeSecs = 60*60*24,
    StoppingRounds = 10,

    # Model args
    Trees = 50L,
    MaxDepth = 20,
    SampleRate = 0.632,
    MTries = -1,
    ColSampleRatePerTree = 1,
    ColSampleRatePerTreeLevel = 1,
    MinRows = 1,
    NBins = 20,
    NBinsCats = 1024,
    NBinsTopLevel = 1024,
    HistogramType = "AUTO",
    CategoricalEncoding = "AUTO")
```

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

AutoH2oDRFHurdleModel   *AutoH2oDRFHurdleModel*

Description

AutoH2oDRFHurdleModel for hurdle modeling

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 = {      gc()
   paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo",
    intern = TRUE))/1e+06)), "G") },
  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 Primary-DateColumn)

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, Validation-Data.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 - Hurdle Modeling: AutoCatBoostHurdleModel(), AutoH2oGBMHurdleModel(), AutoLightGBMHurdleModel(), 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 = {gc();paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo", inter
  NThreads = max(1L, parallel::detectCores()-2L),
  Trees = 1000L,
  GridTune = FALSE,
  MaxModelsInGrid = 1L,
  NumOfParDepPlots = 10L,
  PassInGrid = NULL)

## End(Not run)
```

AutoH2oDRFMultiClass   *AutoH2oDRFMultiClass*

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(
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
  data = NULL,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  WeightsColumn = NULL,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  IfSaveModel = "mojo",
  MaxMem = {      gc()
   paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo",
     intern = TRUE))/1e+06)), "G") },
  NThreads = max(1, parallel::detectCores() - 2),
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
```

```
    H2OShutdown = FALSE,
    H2OStartUp = TRUE,
    DebugMode = FALSE,
    eval_metric = "logloss",
    GridTune = FALSE,
    GridStrategy = "RandomDiscrete",
    MaxRunTimeSecs = 60 * 60 * 24,
    StoppingRounds = 10,
    MaxModelsInGrid = 2,
    Trees = 50,
    MaxDepth = 20L,
    SampleRate = 0.632,
    MTries = -1,
    ColSampleRatePerTree = 1,
    ColSampleRatePerTreeLevel = 1,
    MinRows = 1,
    NBins = 20,
    NBinsCats = 1024,
    NBinsTopLevel = 1024,
    HistogramType = "AUTO",
    CategoricalEncoding = "AUTO"
)
```

Arguments

OutputSelection

> You can select what type of output you want returned. Choose from c("EvalMetrics", "PDFs", "Score_TrainData")

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)

WeightsColumn   Column name of a weights column

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

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
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
H2OShutdown	Set to TRUE to have H2O shutdown after running this function
H2OStartUp	Defaults to TRUE which means H2O will be started inside the function
DebugMode	Set to TRUE to print steps to screen
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.
GridStrategy	Default "Cartesian"
MaxRunTimeSecs	Default 86400
StoppingRounds	Default 10
MaxModelsInGrid	
	Number of models to test from grid options (1080 total possible options)
Trees	The maximum number of trees you want in your models
MaxDepth	Default 20
SampleRate	Default 0.632
MTries	Default -1 means it will default to number of features divided by 3
ColSampleRatePerTree	
	Default 1
ColSampleRatePerTreeLevel	
	Default 1
MinRows	Default 1
NBins	Default 20
NBinsCats	Default 1024
NBinsTopLevel	Default 1024
HistogramType	Default "AUTO"
CategoricalEncoding	
	Default "AUTO"

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(), 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(
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
  data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = "Adrian",
  FeatureColNames = names(data)[!names(data) %in% c("IDcol_1", "IDcol_2","Adrian")],
  WeightsColumn = NULL,
  eval_metric = "logloss",
 MaxMem = {gc();paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo", inter
  NThreads = max(1, parallel::detectCores()-2),
  model_path = normalizePath("./"),
  metadata_path = file.path(normalizePath("./")),
  ModelID = "FirstModel",
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = FALSE,
  H2OStartUp = TRUE,
  DebugMode = FALSE,

  # Grid Tuning Args
  GridStrategy = "RandomDiscrete",
  GridTune = FALSE,
  MaxModelsInGrid = 10,
  MaxRunTimeSecs = 60*60*24,
  StoppingRounds = 10,

  # ML args
  Trees = 50,
  MaxDepth = 20,
  SampleRate = 0.632,
  MTries = -1,
  ColSampleRatePerTree = 1,
  ColSampleRatePerTreeLevel = 1,
  MinRows = 1,
  NBins = 20,
  NBinsCats = 1024,
  NBinsTopLevel = 1024,
  HistogramType = "AUTO",
  CategoricalEncoding = "AUTO")
```

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

AutoH2oDRFRegression   *AutoH2oDRFRegression*

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(
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
  data = NULL,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  WeightsColumn = NULL,
  MaxMem = {      gc()
 paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo",
   intern = TRUE))/1e+06)), "G") },
  NThreads = max(1, parallel::detectCores() - 2),
  H2OShutdown = TRUE,
  H2OStartUp = TRUE,
  DebugMode = FALSE,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  SaveInfoToPDF = FALSE,
  IfSaveModel = "mojo",
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  TransformNumericColumns = NULL,
  Methods = c("BoxCox", "Asinh", "Log", "LogPlus1", "Sqrt", "Asin", "Logit"),
  NumOfParDepPlots = 3,
  eval_metric = "RMSE",
  GridTune = FALSE,
  GridStrategy = "RandomDiscrete",
  MaxRunTimeSecs = 60 * 60 * 24,
  StoppingRounds = 10,
  MaxModelsInGrid = 2,
  Trees = 50,
  MaxDepth = 20,
```

```
    SampleRate = 0.632,
    MTries = -1,
    ColSampleRatePerTree = 1,
    ColSampleRatePerTreeLevel = 1,
    MinRows = 1,
    NBins = 20,
    NBinsCats = 1024,
    NBinsTopLevel = 1024,
    HistogramType = "AUTO",
    CategoricalEncoding = "AUTO"
)
```

Arguments

`OutputSelection`	You can select what type of output you want returned. Choose from c("EvalMetrics", "PDFs", "Score_TrainData")
`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)
`WeightsColumn`	Column name of a weights column
`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
`H2OShutdown`	Set to TRUE to shutdown H2O inside the function
`H2OStartUp`	Defaults to TRUE which means H2O will be started inside the function
`DebugMode`	Set to TRUE to print steps to screen
`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
`SaveInfoToPDF`	Set to TRUE to save insights to PDF
`IfSaveModel`	Set to "mojo" to save a mojo file, otherwise "standard" to save a regular H2O model object
`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

TransformNumericColumns

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

Methods   Choose from "YeoJohnson", "BoxCox", "Asinh", "Log", "LogPlus1", "Sqrt", "Asin", or "Logit". If more than one is selected, the one with the best normalization pearson statistic will be used. Identity is automatically selected and compared.

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)

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.

GridStrategy   Default "Cartesian"

MaxRunTimeSecs Default 86400

StoppingRounds Default 10

MaxModelsInGrid

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

Trees   The maximum number of trees you want in your models

MaxDepth   Default 20

SampleRate   Default 0.632

MTries   Default -1 means it will default to number of features divided by 3

ColSampleRatePerTree

        Default 1

ColSampleRatePerTreeLevel

        Default 1

MinRows   Default 1

NBins   Default 20

NBinsCats   Default 1024

NBinsTopLevel   Default 1024

HistogramType Default "AUTO". Select from AUTO", "UniformAdaptive", "Random", "QuantilesGlobal", "RoundRobin"

CategoricalEncoding

        Default "AUTO"

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()`, `AutoLightGBMRegression()`, `AutoNLS()`, `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 = {gc();paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo", int
   NThreads = max(1L, parallel::detectCores() - 2L),
   H2OShutdown = TRUE,
   H2OStartUp = TRUE,
   IfSaveModel = "mojo",

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

   # Metadata arguments
   OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
   model_path = normalizePath("./"),
   metadata_path = NULL,
   ModelID = "FirstModel",
   ReturnModelObjects = TRUE,
   SaveModelObjects = FALSE,
   SaveInfoToPDF = FALSE,
   DebugMode = FALSE,

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

   # Grid Tuning Args
```

```
      GridStrategy = "RandomDiscrete",
      GridTune = FALSE,
      MaxModelsInGrid = 10,
      MaxRunTimeSecs = 60*60*24,
      StoppingRounds = 10,

      # ML Args
      Trees = 50,
      MaxDepth = 20,
      SampleRate = 0.632,
      MTries = -1,
      ColSampleRatePerTree = 1,
      ColSampleRatePerTreeLevel = 1,
      MinRows = 1,
      NBins = 20,
      NBinsCats = 1024,
      NBinsTopLevel = 1024,
      HistogramType = "AUTO",
      CategoricalEncoding = "AUTO")

  ## End(Not run)
```

AutoH2oGAMClassifier   *AutoH2oGAMClassifier*

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(
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
  data = NULL,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  WeightsColumn = NULL,
  GamColNames = NULL,
  Distribution = "binomial",
  Link = "logit",
  eval_metric = "auc",
  CostMatrixWeights = c(1, 0, 0, 1),
  MaxMem = {      gc()
   paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo",
```

```
        intern = TRUE))/1e+06)), "G") },
    NThreads = max(1, parallel::detectCores() - 2),
    model_path = NULL,
    metadata_path = NULL,
    ModelID = "FirstModel",
    NumOfParDepPlots = 3,
    ReturnModelObjects = TRUE,
    SaveModelObjects = FALSE,
    SaveInfoToPDF = FALSE,
    IfSaveModel = "mojo",
    H2OShutdown = FALSE,
    H2OStartUp = TRUE,
    DebugMode = FALSE,
    GridTune = FALSE,
    GridStrategy = "Cartesian",
    StoppingRounds = 10,
    MaxRunTimeSecs = 3600 * 24 * 7,
    MaxModelsInGrid = 2,
    num_knots = NULL,
    keep_gam_cols = TRUE,
    Solver = "AUTO",
    Alpha = 0.5,
    Lambda = NULL,
    LambdaSearch = FALSE,
    NLambdas = -1,
    Standardize = TRUE,
    RemoveCollinearColumns = FALSE,
    InterceptInclude = TRUE,
    NonNegativeCoefficients = FALSE
)
```

Arguments

`OutputSelection`

You can select what type of output you want returned. Choose from c("EvalMetrics", "PDFs", "Score_TrainData")

`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)

`WeightsColumn`　Weighted classification

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"
CostMatrixWeights	
	A vector with 4 elements c(True Positive Cost, False Negative Cost, False Positive Cost, True Negative Cost). Default c(1,0,0,1),
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
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
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
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
H2OStartUp	Set to TRUE to start up H2O inside function
DebugMode	Set to TRUE to get a print out of steps taken internally
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.
GridStrategy	"RandomDiscrete" or "Cartesian"
StoppingRounds	Iterations in grid tuning
MaxRunTimeSecs	Max run time in seconds
MaxModelsInGrid	
	Number of models to test from grid options (1080 total possible options)
num_knots	Numeric values for gam
keep_gam_cols	Logical
Solver	Default "AUTO". Options include "IRLSM", "L_BFGS", "COORDINATE_DESCENT_NAIVE", "COORDINATE_DESCENT", "GRADIENT_DESCENT_LH", "GRADIENT_DESCENT_SQERR
Alpha	Gridable. Default 0.5 Otherwise supply a value between 0 and 1. 1 is equivalent to Lasso regression. 0 is equivalent to Ridge regression. Inbetween for a blend of the two.
Lambda	Gridable. Default NULL. Regularization strength.
LambdaSearch	Default FALSE.

```
NLambdas          Default -1
Standardize       Default TRUE. Standardize numerical columns
RemoveCollinearColumns
                  Default FALSE. Removes some of the linearly dependent columns
InterceptInclude
                  Default TRUE
NonNegativeCoefficients
                  Default 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(), AutoH2oGBMClassifier(), AutoH2oGLMClassifier(), AutoH2oMLClassifier(), AutoLightGBMClassifier(), 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 = {gc();paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo", inter
  NThreads = max(1, parallel::detectCores()-2),
  H2OShutdown = TRUE,
  H2OStartUp = TRUE,
  IfSaveModel = "mojo",

  # Model evaluation args
  CostMatrixWeights = c(1,0,0,1),
```

```
eval_metric = "auc",
NumOfParDepPlots = 3,

# Metadata arguments
OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
model_path = NULL,
metadata_path = NULL,
ModelID = "FirstModel",
ReturnModelObjects = TRUE,
SaveModelObjects = FALSE,
SaveInfoToPDF = FALSE,
DebugMode = FALSE,

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

# ML args
num_knots = NULL,
keep_gam_cols = TRUE,
GridTune = FALSE,
GridStrategy = "Cartesian",
StoppingRounds = 10,
MaxRunTimeSecs = 3600 * 24 * 7,
MaxModelsInGrid = 10,
Distribution = "binomial",
Link = "logit",
Solver = "AUTO",
Alpha = 0.5,
Lambda = NULL,
LambdaSearch = FALSE,
NLambdas = -1,
Standardize = TRUE,
RemoveCollinearColumns = FALSE,
InterceptInclude = TRUE,
NonNegativeCoefficients = FALSE)
```

AutoH2oGAMMultiClass   *AutoH2oGAMMultiClass*

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(
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
  data = NULL,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  WeightsColumn = NULL,
  GamColNames = NULL,
  eval_metric = "logloss",
  MaxMem = {      gc()
 paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo",
   intern = TRUE))/1e+06)), "G") },
  NThreads = max(1, parallel::detectCores() - 2),
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = FALSE,
  H2OStartUp = TRUE,
  DebugMode = FALSE,
  GridTune = FALSE,
  GridStrategy = "Cartesian",
  StoppingRounds = 10,
  MaxRunTimeSecs = 3600 * 24 * 7,
  MaxModelsInGrid = 2,
  Distribution = "multinomial",
  Link = "Family_Default",
  num_knots = NULL,
  keep_gam_cols = TRUE,
  Solver = "AUTO",
  Alpha = 0.5,
  Lambda = NULL,
  LambdaSearch = FALSE,
  NLambdas = -1,
  Standardize = TRUE,
  RemoveCollinearColumns = FALSE,
  InterceptInclude = TRUE,
  NonNegativeCoefficients = FALSE
)
```

Arguments

OutputSelection
> You can select what type of output you want returned. Choose from c("EvalMetrics", "PDFs", "Score_TrainData")

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)
WeightsColumn	Weighted classification
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"
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
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
H2OStartUp	Set to TRUE to start up H2O inside function
DebugMode	Set to TRUE to print steps to screen
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.
GridStrategy	"RandomDiscrete" or "Cartesian"
StoppingRounds	Iterations in grid tuning
MaxRunTimeSecs	Max run time in seconds
MaxModelsInGrid	
	Number of models to test from grid options (1080 total possible options)
num_knots	Numeric values for gam
keep_gam_cols	Logical
Solver	Default "AUTO". Options include "IRLSM", "L_BFGS", "COORDINATE_DESCENT_NAIVE", "COORDINATE_DESCENT", "GRADIENT_DESCENT_LH", "GRADIENT_DESCENT_SQERR"
Alpha	Gridable. Default 0.5 Otherwise supply a value between 0 and 1. 1 is equivalent to Lasso regression. 0 is equivalent to Ridge regression. Inbetween for a blend of the two.

Lambda	Gridable. Default NULL. Regularization strength.
LambdaSearch	Default FALSE.
NLambdas	Default -1
Standardize	Default TRUE. Standardize numerical columns
RemoveCollinearColumns	
	Default FALSE. Removes some of the linearly dependent columns
InterceptInclude	
	Default TRUE
NonNegativeCoefficients	
	Default 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`(), `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(
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
  data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = "Adrian",
  FeatureColNames = names(data)[!names(data) %in% c("IDcol_1", "IDcol_2","Adrian")],
  WeightsColumn = NULL,
  GamColNames = GamCols,
```

```
  eval_metric = "logloss",
 MaxMem = {gc();paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo", inter
 NThreads = max(1, parallel::detectCores()-2),
 model_path = normalizePath("./"),
 metadata_path = NULL,
 ModelID = "FirstModel",
 ReturnModelObjects = TRUE,
 SaveModelObjects = FALSE,
 IfSaveModel = "mojo",
 H2OShutdown = FALSE,
 H2OStartUp = TRUE,
 DebugMode = FALSE,

 # ML args
 num_knots = NULL,
 keep_gam_cols = TRUE,
 GridTune = FALSE,
 GridStrategy = "Cartesian",
 StoppingRounds = 10,
 MaxRunTimeSecs = 3600 * 24 * 7,
 MaxModelsInGrid = 10,
 Distribution = "multinomial",
 Link = "Family_Default",
 Solver = "AUTO",
 Alpha = 0.5,
 Lambda = NULL,
 LambdaSearch = FALSE,
 NLambdas = -1,
 Standardize = TRUE,
 RemoveCollinearColumns = FALSE,
 InterceptInclude = TRUE,
 NonNegativeCoefficients = FALSE)
```

AutoH2oGAMRegression   *AutoH2oGAMRegression*

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(
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
  data = NULL,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
```

```
      TargetColumnName = NULL,
      FeatureColNames = NULL,
      InteractionColNumbers = NULL,
      WeightsColumn = NULL,
      GamColNames = NULL,
      Distribution = "gaussian",
      Link = "identity",
      TweedieLinkPower = NULL,
      TweedieVariancePower = NULL,
      TransformNumericColumns = NULL,
      Methods = c("BoxCox", "Asinh", "Log", "LogPlus1", "Sqrt", "Asin", "Logit"),
      eval_metric = "RMSE",
      MaxMem = {      gc()
       paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo",
         intern = TRUE))/1e+06)), "G") },
      NThreads = max(1, parallel::detectCores() - 2),
      model_path = NULL,
      metadata_path = NULL,
      ModelID = "FirstModel",
      NumOfParDepPlots = 3,
      ReturnModelObjects = TRUE,
      SaveModelObjects = FALSE,
      SaveInfoToPDF = FALSE,
      IfSaveModel = "mojo",
      H2OShutdown = TRUE,
      H2OStartUp = TRUE,
      GridTune = FALSE,
      GridStrategy = "Cartesian",
      StoppingRounds = 10,
      MaxRunTimeSecs = 3600 * 24 * 7,
      MaxModelsInGrid = 2,
      num_knots = NULL,
      keep_gam_cols = TRUE,
      Solver = "AUTO",
      Alpha = 0.5,
      Lambda = NULL,
      LambdaSearch = FALSE,
      NLambdas = -1,
      Standardize = TRUE,
      RemoveCollinearColumns = FALSE,
      InterceptInclude = TRUE,
      NonNegativeCoefficients = FALSE,
      DebugMode = FALSE
    )
```

Arguments

`OutputSelection`

> You can select what type of output you want returned. Choose from c("EvalMetrics",
> "PDFs", "Score_TrainData")

`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)
InteractionColNumbers	
	Column numbers of the features you want to be pairwise interacted
WeightsColumn	Column name of a weights column
GamColNames	GAM column names. Up to 9 features
Distribution	: "AUTO", "gaussian", "binomial", "quasi-binomial", "ordinal", "multinomial", "poisson", "gamma", "tweedie", "negative-binomial", "fractionalbinomial"
Link	"family_default", "identity", "logit", "log", "inverse", "tweedie","ologit"
TweedieLinkPower	
	See h2o docs for background
TweedieVariancePower	
	See h2o docs for background
TransformNumericColumns	
	Set to NULL to do nothing; otherwise supply the column names of numeric variables you want transformed
Methods	Choose from "BoxCox", "Asinh", "Log", "LogPlus1", "Sqrt", "Asin", or "Logit". If more than one is selected, the one with the best normalization pearson statistic will be used. Identity is automatically selected and compared.
eval_metric	This is the metric used to identify best grid tuned model. Choose from "MSE", "RMSE", "MAE", "RMSLE"
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
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
SaveInfoToPDF	Set to TRUE to save insights to PDF

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 inside the function
H2OStartUp	Defaults to TRUE which means H2O will be started inside the function
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.
GridStrategy	"RandomDiscrete" or "Cartesian"
StoppingRounds	Iterations in grid tuning
MaxRunTimeSecs	Max run time in seconds
MaxModelsInGrid	
	Number of models to test from grid options (1080 total possible options)
num_knots	Numeric values for gam
keep_gam_cols	Logical
Solver	Default "AUTO". Options include "IRLSM", "L_BFGS", "COORDINATE_DESCENT_NAIVE", "COORDINATE_DESCENT", "GRADIENT_DESCENT_LH", "GRADIENT_DESCENT_SQERR
Alpha	Gridable. Default 0.5 Otherwise supply a value between 0 and 1. 1 is equivalent to Lasso regression. 0 is equivalent to Ridge regression. Inbetween for a blend of the two.
Lambda	Gridable. Default NULL. Regularization strength.
LambdaSearch	Default FALSE.
NLambdas	Default -1
Standardize	Default TRUE. Standardize numerical columns
RemoveCollinearColumns	
	Default FALSE. Removes some of the linearly dependent columns
InterceptInclude	
	Default TRUE
NonNegativeCoefficients	
	Default FALSE
DebugMode	Set to TRUE to get a printout of steps taken

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()`, `AutoH2oGBMRegression()`, `AutoH2oGLMRegression()`, `AutoH2oMLRegression()`, `AutoLightGBMRegression()`, `AutoNLS()`, `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 = {gc();paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo", inter
 NThreads = max(1, parallel::detectCores()-2),
 H2OShutdown = TRUE,
 H2OStartUp = TRUE,
 IfSaveModel = "mojo",

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

 # Metadata arguments
 OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
 model_path = NULL,
 metadata_path = NULL,
 ModelID = "FirstModel",
 ReturnModelObjects = TRUE,
 SaveModelObjects = FALSE,
 SaveInfoToPDF = FALSE,

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

 # Model args
```

```
num_knots = NULL,
keep_gam_cols = TRUE,
GridTune = FALSE,
GridStrategy = "Cartesian",
StoppingRounds = 10,
MaxRunTimeSecs = 3600 * 24 * 7,
MaxModelsInGrid = 10,
Distribution = "gaussian",
Link = "Family_Default",
TweedieLinkPower = NULL,
TweedieVariancePower = NULL,
Solver = "AUTO",
Alpha = 0.5,
Lambda = NULL,
LambdaSearch = FALSE,
NLambdas = -1,
Standardize = TRUE,
RemoveCollinearColumns = FALSE,
InterceptInclude = TRUE,
NonNegativeCoefficients = FALSE,
DebugMode = FALSE)
```

AutoH2oGBMClassifier   *AutoH2oGBMClassifier*

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(
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
  data = NULL,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  WeightsColumn = NULL,
  MaxMem = {      gc()
   paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo",
    intern = TRUE))/1e+06)), "G") },
  NThreads = max(1L, parallel::detectCores() - 2L),
  model_path = NULL,
  metadata_path = NULL,
```

```
  ModelID = "FirstModel",
  NumOfParDepPlots = 3L,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  SaveInfoToPDF = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = FALSE,
  H2OStartUp = TRUE,
  DebugMode = FALSE,
  GridStrategy = "Cartesian",
  MaxRunTimeSecs = 60 * 60 * 24,
  StoppingRounds = 10,
  MaxModelsInGrid = 2,
  eval_metric = "auc",
  CostMatrixWeights = c(1, 0, 0, 1),
  Trees = 50L,
  GridTune = FALSE,
  LearnRate = 0.1,
  LearnRateAnnealing = 1,
  Distribution = "bernoulli",
  MaxDepth = 20,
  SampleRate = 0.632,
  ColSampleRate = 1,
  ColSampleRatePerTree = 1,
  ColSampleRatePerTreeLevel = 1,
  MinRows = 1,
  NBins = 20,
  NBinsCats = 1024,
  NBinsTopLevel = 1024,
  HistogramType = "AUTO",
  CategoricalEncoding = "AUTO"
)
```

Arguments

OutputSelection

You can select what type of output you want returned. Choose from c("EvalMetrics", "PDFs", "Score_TrainData")

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)

WeightsColumn	Column name of a weights column
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
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
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
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 inside the function
H2OStartUp	Defaults to TRUE which means H2O will be started inside the function
DebugMode	Set to TRUE to get a printout of the steps taken internally
GridStrategy	Default "Cartesian"
MaxRunTimeSecs	Default 60*60*24
StoppingRounds	Number of runs
MaxModelsInGrid	
	Number of models to test from grid options (1080 total possible options)
eval_metric	This is the metric used to identify best grid tuned model. Choose from "auc","logloss","aucpr", "lift_top_group","misclassification","mean_per_class_error"
CostMatrixWeights	
	A vector with 4 elements c(True Positive Cost, False Negative Cost, False Positive Cost, True Negative Cost). Default c(1,0,0,1),
Trees	The maximum number of trees you want in your models
GridTune	Set to TRUE to run a grid tuning procedure. Set a number in MaxModelsInGrid to tell the procedure how many models you want to test.
LearnRate	Default 0.10
LearnRateAnnealing	
	Default 1
Distribution	Choose from "AUTO", "bernoulli", and "quasibinomial"
MaxDepth	Default 20
SampleRate	Default 0.632
ColSampleRate	Default 1
ColSampleRatePerTree	
	Default 1

```
ColSampleRatePerTreeLevel
                  Default 1
MinRows           Default 1
NBins             Default 20
NBinsCats         Default 1024
NBinsTopLevel     Default 1024
HistogramType     Default "AUTO"
CategoricalEncoding
                  Default "AUTO"
```

Value

Saves to file and returned in list: VariableImportance.csv, Model, ValidationData.csv, Evaluation-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(), AutoLightGBMClassifier(), 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 = {gc();paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo", int
    NThreads = max(1, parallel::detectCores()-2),
    H2OShutdown = TRUE,
    H2OStartUp = TRUE,
    IfSaveModel = "mojo",

    # Model evaluation
    CostMatrixWeights = c(1,0,0,1),
    eval_metric = "auc",
    NumOfParDepPlots = 3,

    # Metadata arguments
```

```
        OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
        model_path = normalizePath("./"),
        metadata_path = file.path(normalizePath("./")),
        ModelID = "FirstModel",
        ReturnModelObjects = TRUE,
        SaveModelObjects = FALSE,
        SaveInfoToPDF = FALSE,
        DebugMode = FALSE,

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

        # ML grid tuning args
        GridTune = FALSE,
        GridStrategy = "Cartesian",
        MaxRunTimeSecs = 60*60*24,
        StoppingRounds = 10,
        MaxModelsInGrid = 2,

        # Model args
        Trees = 50,
        LearnRate = 0.10,
        LearnRateAnnealing = 1,
        Distribution = "bernoulli",
        MaxDepth = 20,
        SampleRate = 0.632,
        ColSampleRate = 1,
        ColSampleRatePerTree = 1,
        ColSampleRatePerTreeLevel  = 1,
        MinRows = 1,
        NBins = 20,
        NBinsCats = 1024,
        NBinsTopLevel = 1024,
        HistogramType = "AUTO",
        CategoricalEncoding = "AUTO")

    ## End(Not run)
```

AutoH2oGBMHurdleModel   *AutoH2oGBMHurdleModel*

Description

AutoH2oGBMHurdleModel for hurdle modeing

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 = {      gc()
         paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo",
           intern = TRUE))/1e+06)), "G") },
        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 Primary-DateColumn)
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, Validation-Data.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 - Hurdle Modeling: `AutoCatBoostHurdleModel`(), `AutoH2oDRFHurdleModel`(), `AutoLightGBMHurdleModel`(), `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),
 MaxMem = {gc();paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo", inter
  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*

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(
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
  data = NULL,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  WeightsColumn = NULL,
  MaxMem = {      gc()
   paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo",
     intern = TRUE))/1e+06)), "G") },
  NThreads = max(1L, parallel::detectCores() - 2L),
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  NumOfParDepPlots = 3L,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = TRUE,
  H2OStartUp = TRUE,
  DebugMode = FALSE,
  GridTune = FALSE,
  GridStrategy = "Cartesian",
  MaxRunTimeSecs = 60 * 60 * 24,
  StoppingRounds = 10,
  MaxModelsInGrid = 2,
  eval_metric = "auc",
  Trees = 50L,
  LearnRate = 0.1,
  LearnRateAnnealing = 1,
  Distribution = "multinomial",
  MaxDepth = 20,
  SampleRate = 0.632,
  MTries = -1,
  ColSampleRate = 1,
```

```
    ColSampleRatePerTree = 1,
    ColSampleRatePerTreeLevel = 1,
    MinRows = 1,
    NBins = 20,
    NBinsCats = 1024,
    NBinsTopLevel = 1024,
    HistogramType = "AUTO",
    CategoricalEncoding = "AUTO"
)
```

Arguments

OutputSelection

You can select what type of output you want returned. Choose from c("EvalMetrics", "PDFs", "Score_TrainData")

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)

WeightsColumn Column name of a weights column

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

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 TRUE to shutdown H2O inside the function

H2OStartUp Defaults to TRUE which means H2O will be started inside the function

DebugMode	Set to TRUE to print steps
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.
GridStrategy	Default "Cartesian"
MaxRunTimeSecs	Default 60*60*24
StoppingRounds	Number of runs
MaxModelsInGrid	
	Number of models to test from grid options (1080 total possible options)
eval_metric	This is the metric used to identify best grid tuned model. Choose from "auc", "logloss"
Trees	The maximum number of trees you want in your models
LearnRate	Default 0.10
LearnRateAnnealing	
	Default 1
Distribution	Choose from "multinomial". Placeholder in more options get added
MaxDepth	Default 20
SampleRate	Default 0.632
ColSampleRate	Default 1
ColSampleRatePerTree	
	Default 1
ColSampleRatePerTreeLevel	
	Default 1
MinRows	Default 1
NBins	Default 20
NBinsCats	Default 1024
NBinsTopLevel	Default 1024
HistogramType	Default "AUTO"
CategoricalEncoding	
	Default "AUTO"
SaveInfoToPDF	Set to TRUE to save insights to PDF

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(
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
  data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = "Adrian",
  FeatureColNames = names(data)[!names(data) %in% c("IDcol_1", "IDcol_2","Adrian")],
  WeightsColumn = NULL,
  eval_metric = "logloss",
 MaxMem = {gc();paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo", inter
  NThreads = max(1, parallel::detectCores()-2),
  model_path = normalizePath("./"),
  metadata_path = file.path(normalizePath("./")),
  ModelID = "FirstModel",
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = TRUE,
  H2OStartUp = TRUE,
  DebugMode = FALSE,

  # Model args
  GridTune = FALSE,
  GridStrategy = "Cartesian",
  MaxRunTimeSecs = 60*60*24,
  StoppingRounds = 10,
  MaxModelsInGrid = 2,
  Trees = 50,
  LearnRate = 0.10,
  LearnRateAnnealing = 1,
  eval_metric = "RMSE",
  Distribution = "multinomial",
  MaxDepth = 20,
  SampleRate = 0.632,
  ColSampleRate = 1,
  ColSampleRatePerTree = 1,
  ColSampleRatePerTreeLevel  = 1,
  MinRows = 1,
  NBins = 20,
  NBinsCats = 1024,
  NBinsTopLevel = 1024,
  HistogramType = "AUTO",
  CategoricalEncoding = "AUTO")
```

AutoH2oGBMRegression   *AutoH2oGBMRegression*

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(
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
  data = NULL,
  TrainOnFull = FALSE,
  ValidationData,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  WeightsColumn = NULL,
  TransformNumericColumns = NULL,
  Methods = c("BoxCox", "Asinh", "Log", "LogPlus1", "Sqrt", "Asin", "Logit"),
  MaxMem = {      gc()
 paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo",
   intern = TRUE))/1e+06)), "G") },
  NThreads = max(1, parallel::detectCores() - 2),
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  NumOfParDepPlots = 3,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  SaveInfoToPDF = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = TRUE,
  H2OStartUp = TRUE,
  DebugMode = FALSE,
  GridTune = FALSE,
  GridStrategy = "Cartesian",
  MaxRunTimeSecs = 60 * 60 * 24,
  StoppingRounds = 10,
  MaxModelsInGrid = 2,
  eval_metric = "RMSE",
  Trees = 50,
  LearnRate = 0.1,
  LearnRateAnnealing = 1,
```

```
    Alpha = NULL,
    Distribution = "poisson",
    MaxDepth = 20,
    SampleRate = 0.632,
    MTries = -1,
    ColSampleRate = 1,
    ColSampleRatePerTree = 1,
    ColSampleRatePerTreeLevel = 1,
    MinRows = 1,
    NBins = 20,
    NBinsCats = 1024,
    NBinsTopLevel = 1024,
    HistogramType = "AUTO",
    CategoricalEncoding = "AUTO"
)
```

Arguments

`OutputSelection`

You can select what type of output you want returned. Choose from c("EvalMetrics", "PDFs", "Score_TrainData")

`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)

`WeightsColumn`   Column name of a weights column

`TransformNumericColumns`

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

`Methods`   Choose from "YeoJohnson", "BoxCox", "Asinh", "Log", "LogPlus1", "Sqrt", "Asin", or "Logit". If more than one is selected, the one with the best normalization pearson statistic will be used. Identity is automatically selected and compared.

`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

`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

SaveInfoToPDF   Set to TRUE to save insights to PDF

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 inside the function

H2OStartUp   Defaults to TRUE which means H2O will be started inside the function

DebugMode   Set to TRUE to print steps to screen

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.

GridStrategy   Default "Cartesian"

MaxRunTimeSecs   Default 60*60*24

StoppingRounds   Number of runs

MaxModelsInGrid

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

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

LearnRate   Default 0.10

LearnRateAnnealing

> Default 1

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"

MaxDepth   Default 20

SampleRate   Default 0.632

ColSampleRate   Default 1

ColSampleRatePerTree

> Default 1

ColSampleRatePerTreeLevel

> Default 1

MinRows   Default 1

NBins   Default 20

NBinsCats   Default 1024

NBinsTopLevel   Default 1024

HistogramType   Default "AUTO"

CategoricalEncoding

> Default "AUTO"

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 metadata

Author(s)

Adrian Antico

See Also

Other Automated Supervised Learning - Regression: `AutoCatBoostRegression()`, `AutoH2oDRFRegression()`, `AutoH2oGAMRegression()`, `AutoH2oGLMRegression()`, `AutoH2oMLRegression()`, `AutoLightGBMRegression()`, `AutoNLS()`, `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 = {gc();paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo", int
  NThreads = max(1, parallel::detectCores()-2),
  H2OShutdown = TRUE,
  H2OStartUp = TRUE,
  IfSaveModel = "mojo",

  # Model evaluation
  NumOfParDepPlots = 3,

  # Metadata arguments
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
  model_path = normalizePath("./"),
  metadata_path = file.path(normalizePath("./")),
  ModelID = "FirstModel",
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  SaveInfoToPDF = FALSE,
  DebugMode = FALSE,

  # Data arguments
  data = data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = "Adrian",
```

```
      FeatureColNames = names(data)[!names(data) %in% c("IDcol_1", "IDcol_2","Adrian")],
      WeightsColumn = NULL,
      TransformNumericColumns = NULL,
    Methods = c("BoxCox", "Asinh", "Asin", "Log", "LogPlus1", "Sqrt", "Logit","YeoJohnson"),

      # ML grid tuning args
      GridTune = FALSE,
      GridStrategy = "Cartesian",
      MaxRunTimeSecs = 60*60*24,
      StoppingRounds = 10,
      MaxModelsInGrid = 2,

      # Model args
      Trees = 50,
      LearnRate = 0.10,
      LearnRateAnnealing = 1,
      eval_metric = "RMSE",
      Alpha = NULL,
      Distribution = "poisson",
      MaxDepth = 20,
      SampleRate = 0.632,
      ColSampleRate = 1,
      ColSampleRatePerTree = 1,
      ColSampleRatePerTreeLevel  = 1,
      MinRows = 1,
      NBins = 20,
      NBinsCats = 1024,
      NBinsTopLevel = 1024,
      HistogramType = "AUTO",
      CategoricalEncoding = "AUTO")
```

AutoH2oGLMClassifier   *AutoH2oGLMClassifier*

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(
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
  data = NULL,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
```

```
    TargetColumnName = NULL,
    FeatureColNames = NULL,
    RandomColNumbers = NULL,
    InteractionColNumbers = NULL,
    WeightsColumn = NULL,
    MaxMem = {       gc()
     paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo",
       intern = TRUE))/1e+06)), "G") },
    NThreads = max(1, parallel::detectCores() - 2),
    ModelID = "FirstModel",
    ReturnModelObjects = TRUE,
    model_path = NULL,
    metadata_path = NULL,
    SaveModelObjects = FALSE,
    SaveInfoToPDF = FALSE,
    IfSaveModel = "mojo",
    H2OShutdown = TRUE,
    H2OStartUp = TRUE,
    DebugMode = FALSE,
    MaxModelsInGrid = 2,
    NumOfParDepPlots = 3,
    GridTune = FALSE,
    GridStrategy = "Cartesian",
    StoppingRounds = 10,
    MaxRunTimeSecs = 3600 * 24 * 7,
    Distribution = "binomial",
    Link = "logit",
    eval_metric = "auc",
    CostMatrixWeights = c(1, 0, 0, 1),
    RandomDistribution = NULL,
    RandomLink = NULL,
    Solver = "AUTO",
    Alpha = 0.5,
    Lambda = NULL,
    LambdaSearch = FALSE,
    NLambdas = -1,
    Standardize = TRUE,
    RemoveCollinearColumns = FALSE,
    InterceptInclude = TRUE,
    NonNegativeCoefficients = FALSE
)
```

Arguments

OutputSelection
> You can select what type of output you want returned. Choose from c("EvalMetrics", "PDFs", "Score_TrainData")

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)
RandomColNumbers	
	Random effects column number indicies
InteractionColNumbers	
	Column numbers of the features you want to be pairwise interacted
WeightsColumn	Column name of a weights column
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
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)
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.
SaveModelObjects	
	Set to TRUE to return all modeling objects to your environment
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
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 inside the function
H2OStartUp	Defaults to TRUE which means H2O will be started inside the function
DebugMode	Set to TRUE to print steps to screen
MaxModelsInGrid	
	Number of models to test from grid options (1080 total possible options)
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)
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.
GridStrategy	"RandomDiscrete" or "Cartesian"
StoppingRounds	Iterations in grid tuning
MaxRunTimeSecs	Max run time in seconds
Distribution	"binomial", "fractionalbinomial", "quasibinomial"
eval_metric	This is the metric used to identify best grid tuned model. Choose from "auc"

CostMatrixWeights

> A vector with 4 elements c(True Positive Cost, False Negative Cost, False Positive Cost, True Negative Cost). Default c(1,0,0,1),

RandomDistribution

> Random effects family. Defaults NULL, otherwise it will run a hierarchical glm

RandomLink   Random effects link. Defaults NULL, otherwise it will run a hierarchical glm

Solver   Default "AUTO". Options include "IRLSM", "L_BFGS", "COORDINATE_DESCENT_NAIVE", "COORDINATE_DESCENT", "GRADIENT_DESCENT_LH", "GRADIENT_DESCENT_SQERR

Alpha   Default 0.5 Otherwise supply a value between 0 and 1. 1 is equivalent to Lasso regression. 0 is equivalent to Ridge regression. Inbetween for a blend of the two.

Lambda   Default NULL. Regularization strength.

LambdaSearch   Default FALSE.

NLambdas   Default -1

Standardize   Default TRUE. Standardize numerical columns

RemoveCollinearColumns

> Default FALSE. Removes some of the linearly dependent columns

InterceptInclude

> Default TRUE

NonNegativeCoefficients

> Default FALSE

link   identity, logit, log, inverse, tweedie

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()`, `AutoLightGBMClassifier()`,
`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(

    # Compute management
  MaxMem = {gc();paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo", int
    NThreads = max(1, parallel::detectCores()-2),
    H2OShutdown = TRUE,
    H2OStartUp = TRUE,
    IfSaveModel = "mojo",

    # Model evaluation args
    CostMatrixWeights = c(1,0,0,1),
    eval_metric = "auc",
    NumOfParDepPlots = 3,

    # Metadata args
    OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
    model_path = NULL,
    metadata_path = NULL,
    ModelID = "FirstModel",
    ReturnModelObjects = TRUE,
    SaveModelObjects = FALSE,
    SaveInfoToPDF = FALSE,
    DebugMode = FALSE,

    # Data args
    data = data,
    TrainOnFull = FALSE,
    ValidationData = NULL,
    TestData = NULL,
    TargetColumnName = "Adrian",
    FeatureColNames = names(data)[!names(data) %in%
      c("IDcol_1", "IDcol_2","Adrian")],
    RandomColNumbers = NULL,
    InteractionColNumbers = NULL,
    WeightsColumn = NULL,

    # ML args
    GridTune = FALSE,
    GridStrategy = "Cartesian",
    StoppingRounds = 10,
    MaxRunTimeSecs = 3600 * 24 * 7,
    MaxModelsInGrid = 10,
    Distribution = "binomial",
    Link = "logit",
    RandomDistribution = NULL,
    RandomLink = NULL,
    Solver = "AUTO",
    Alpha = 0.5,
    Lambda = NULL,
    LambdaSearch = FALSE,
    NLambdas = -1,
    Standardize = TRUE,
    RemoveCollinearColumns = FALSE,
    InterceptInclude = TRUE,
    NonNegativeCoefficients = FALSE)
```

AutoH2oGLMMultiClass   *AutoH2oGLMMultiClass*

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(
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
  data = NULL,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  RandomColNumbers = NULL,
  InteractionColNumbers = NULL,
  WeightsColumn = NULL,
  MaxMem = {     gc()
   paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo",
     intern = TRUE))/1e+06)), "G") },
  NThreads = max(1, parallel::detectCores() - 2),
  ModelID = "FirstModel",
  ReturnModelObjects = TRUE,
  model_path = NULL,
  metadata_path = NULL,
  DebugMode = FALSE,
  SaveModelObjects = FALSE,
  SaveInfoToPDF = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = TRUE,
  H2OStartUp = TRUE,
  MaxModelsInGrid = 2,
  NumOfParDepPlots = 3,
  GridTune = FALSE,
  GridStrategy = "Cartesian",
  StoppingRounds = 10,
  MaxRunTimeSecs = 3600 * 24 * 7,
  Distribution = "multinomial",
  Link = "family_default",
  eval_metric = "logloss",
  RandomDistribution = NULL,
  RandomLink = NULL,
  Solver = "AUTO",
```

```
  Alpha = 0.5,
  Lambda = NULL,
  LambdaSearch = FALSE,
  NLambdas = -1,
  Standardize = TRUE,
  RemoveCollinearColumns = FALSE,
  InterceptInclude = TRUE,
  NonNegativeCoefficients = FALSE
)
```

Arguments

OutputSelection
                  You can select what type of output you want returned. Choose from c("EvalMetrics",
                  "PDFs", "Score_TrainData")

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 hyperparame-
                  ters.

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)

RandomColNumbers
                  Random effects column number indicies

InteractionColNumbers
                  Column numbers of the features you want to be pairwise interacted

WeightsColumn   Column name of a weights column

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

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)

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.

DebugMode   Set to TRUE to see a printout of each step

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

SaveInfoToPDF   Set to TRUE to save insights to PDF

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 inside the function |
| H2OStartUp | Defaults to TRUE which means H2O will be started inside the function |
| MaxModelsInGrid | |
| | Number of models to test from grid options (1080 total possible options) |
| 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) |
| 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. |
| GridStrategy | "RandomDiscrete" or "Cartesian" |
| StoppingRounds | Iterations in grid tuning |
| MaxRunTimeSecs | Max run time in seconds |
| Distribution | "multinomial" |
| eval_metric | This is the metric used to identify best grid tuned model. Choose from "logloss" |
| RandomDistribution | |
| | Random effects family. Defaults NULL, otherwise it will run a hierarchical glm |
| RandomLink | Random effects link. Defaults NULL, otherwise it will run a hierarchical glm |
| Solver | Default "AUTO". Options include "IRLSM", "L_BFGS", "COORDINATE_DESCENT_NAIVE", "COORDINATE_DESCENT", "GRADIENT_DESCENT_LH", "GRADIENT_DESCENT_SQERR" |
| Alpha | Default 0.5 Otherwise supply a value between 0 and 1. 1 is equivalent to Lasso regression. 0 is equivalent to Ridge regression. Inbetween for a blend of the two. |
| Lambda | Default NULL. Regularization strength. |
| LambdaSearch | Default FALSE. |
| NLambdas | Default -1 |
| Standardize | Default TRUE. Standardize numerical columns |
| RemoveCollinearColumns | |
| | Default FALSE. Removes some of the linearly dependent columns |
| InterceptInclude | |
| | Default TRUE |
| NonNegativeCoefficients | |
| | Default FALSE |
| link | "family_default" |

## 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()`, `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(

  # Compute management
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
 MaxMem = {gc();paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo", inter
  NThreads = max(1, parallel::detectCores()-2),
  H2OShutdown = TRUE,
  H2OStartUp = TRUE,
  IfSaveModel = "mojo",

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

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

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

  # Model args
  GridTune = FALSE,
  GridStrategy = "Cartesian",
  StoppingRounds = 10,
  MaxRunTimeSecs = 3600 * 24 * 7,
  MaxModelsInGrid = 10,
  Distribution = "multinomial",
  Link = "family_default",
  RandomDistribution = NULL,
  RandomLink = NULL,
```

```
  Solver = "AUTO",
  Alpha = 0.5,
  Lambda = NULL,
  LambdaSearch = FALSE,
  NLambdas = -1,
  Standardize = TRUE,
  RemoveCollinearColumns = FALSE,
  InterceptInclude = TRUE,
  NonNegativeCoefficients = FALSE)
```

AutoH2oGLMRegression   *AutoH2oGLMRegression*

Description

AutoH2oGLMis 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(
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
  data = NULL,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  RandomColNumbers = NULL,
  InteractionColNumbers = NULL,
  WeightsColumn = NULL,
  MaxMem = {      gc()
  paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo",
    intern = TRUE))/1e+06)), "G") },
  NThreads = max(1, parallel::detectCores() - 2),
  ModelID = "FirstModel",
  ReturnModelObjects = TRUE,
  model_path = NULL,
  metadata_path = NULL,
  SaveModelObjects = FALSE,
  SaveInfoToPDF = FALSE,
  IfSaveModel = "mojo",
  H2OShutdown = TRUE,
  H2OStartUp = TRUE,
  DebugMode = FALSE,
  TransformNumericColumns = NULL,
  Methods = c("BoxCox", "Asinh", "Log", "LogPlus1", "Sqrt", "Asin", "Logit"),
```

```
    NumOfParDepPlots = 3,
    GridTune = FALSE,
    GridStrategy = "Cartesian",
    StoppingRounds = 10,
    MaxRunTimeSecs = 3600 * 24 * 7,
    MaxModelsInGrid = 2,
    Distribution = "gaussian",
    Link = "identity",
    TweedieLinkPower = NULL,
    TweedieVariancePower = NULL,
    eval_metric = "RMSE",
    RandomDistribution = NULL,
    RandomLink = NULL,
    Solver = "AUTO",
    Alpha = 0.5,
    Lambda = NULL,
    LambdaSearch = FALSE,
    NLambdas = -1,
    Standardize = TRUE,
    RemoveCollinearColumns = FALSE,
    InterceptInclude = TRUE,
    NonNegativeCoefficients = FALSE
)
```

Arguments

OutputSelection

You can select what type of output you want returned. Choose from c("EvalMetrics", "PDFs", "Score_TrainData")

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)

RandomColNumbers

Random effects column number indicies

InteractionColNumbers

Column numbers of the features you want to be pairwise interacted

WeightsColumn   Column name of a weights column

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

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)
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.
SaveModelObjects	
	Set to TRUE to return all modeling objects to your environment
SaveInfoToPDF	Set to TRUE to save insights to PDF
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 inside the function
H2OStartUp	Defaults to TRUE which means H2O will be started inside the function
DebugMode	Set to TRUE to print out steps to screen
TransformNumericColumns	
	Set to NULL to do nothing; otherwise supply the column names of numeric variables you want transformed
Methods	Choose from "YeoJohnson", "BoxCox", "Asinh", "Log", "LogPlus1", "Sqrt", "Asin", or "Logit". If more than one is selected, the one with the best normalization pearson statistic will be used. Identity is automatically selected and compared.
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)
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.
GridStrategy	"RandomDiscrete" or "Cartesian"
StoppingRounds	Iterations in grid tuning
MaxRunTimeSecs	Max run time in seconds
MaxModelsInGrid	
	Number of models to test from grid options (1080 total possible options)
Distribution	"AUTO", "gaussian", "poisson", "gamma", "tweedie", "negativebinomial"
Link	"family_default", "identity", "log", "inverse", "tweedie"
TweedieLinkPower	
	See h2o docs for background
TweedieVariancePower	
	See h2o docs for background
eval_metric	This is the metric used to identify best grid tuned model. Choose from "MSE", "RMSE", "MAE", "RMSLE"
RandomDistribution	
	Random effects family. Defaults NULL, otherwise it will run a hierarchical glm
RandomLink	Random effects link. Defaults NULL, otherwise it will run a hierarchical glm
Solver	Default "AUTO". Options include "IRLSM", "L_BFGS", "COORDINATE_DESCENT_NAIVE", "COORDINATE_DESCENT", "GRADIENT_DESCENT_LH", "GRADIENT_DESCENT_SQERR

Alpha	Default 0.5 Otherwise supply a value between 0 and 1. 1 is equivalent to Lasso regression. 0 is equivalent to Ridge regression. Inbetween for a blend of the two.
Lambda	Default NULL. Regularization strength.
LambdaSearch	Default FALSE.
NLambdas	Default -1
Standardize	Default TRUE. Standardize numerical columns
RemoveCollinearColumns	
	Default FALSE. Removes some of the linearly dependent columns
InterceptInclude	
	Default TRUE
NonNegativeCoefficients	
	Default 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(), AutoLightGBMRegression(), AutoNLS(), 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 = {gc();paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo", int
    NThreads = max(1, parallel::detectCores()-2),
    H2OShutdown = TRUE,
    H2OStartUp = TRUE,
    IfSaveModel = "mojo",

    # Model evaluation:
```

```
    eval_metric = "RMSE",
    NumOfParDepPlots = 3,

    # Metadata arguments
    OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
    model_path = NULL,
    metadata_path = NULL,
    ModelID = "FirstModel",
    ReturnModelObjects = TRUE,
    SaveModelObjects = FALSE,
    SaveInfoToPDF = FALSE,
    DebugMode = FALSE,

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

    # Model args
    GridTune = FALSE,
    GridStrategy = "Cartesian",
    StoppingRounds = 10,
    MaxRunTimeSecs = 3600 * 24 * 7,
    MaxModelsInGrid = 10,
    Distribution = "gaussian",
    Link = "identity",
    TweedieLinkPower = NULL,
    TweedieVariancePower = NULL,
    RandomDistribution = NULL,
    RandomLink = NULL,
    Solver = "AUTO",
    Alpha = 0.5,
    Lambda = NULL,
    LambdaSearch = FALSE,
    NLambdas = -1,
    Standardize = TRUE,
    RemoveCollinearColumns = FALSE,
    InterceptInclude = TRUE,
    NonNegativeCoefficients = FALSE)
```

AutoH2oMLClassifier   *AutoH2oMLClassifier*

Description

AutoH2oMLClassifier is an automated H2O modeling framework with grid-tuning and model eval-
uation 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(
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
  data = NULL,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  ExcludeAlgos = NULL,
  eval_metric = "auc",
  CostMatrixWeights = c(1, 0, 0, 1),
  MaxMem = {      gc()
   paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo",
     intern = TRUE))/1e+06)), "G") },
  NThreads = max(1, parallel::detectCores() - 2),
  MaxModelsInGrid = 2,
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  NumOfParDepPlots = 3,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  SaveInfoToPDF = TRUE,
  IfSaveModel = "mojo",
  H2OShutdown = TRUE,
  H2OStartUp = TRUE,
  DebugMode = FALSE
)
```

Arguments

OutputSelection
              You can select what type of output you want returned. Choose from c("EvalMetrics",
              "PDFs", "Score_TrainData")

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 hyperparame-
              ters.

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"

CostMatrixWeights

        A vector with 4 elements c(True Positive Cost, False Negative Cost, False Positive Cost, True Negative Cost). Default c(1,0,0,1),

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

SaveInfoToPDF   Set to TRUE to print model insights to PDF

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

H2OStartUp   Set to FALSE

DebugMode   Set to TRUE to print out steps taken

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()`, `AutoLightGBMClassifier()`, `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(
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
  data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = "Adrian",
  FeatureColNames = names(data)[!names(data) %in% c("IDcol_1", "IDcol_2","Adrian")],
  ExcludeAlgos = NULL,
  eval_metric = "auc",
  CostMatrixWeights = c(1,0,0,1),
 MaxMem = {gc();paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo", inter
  NThreads = max(1, parallel::detectCores()-2),
  MaxModelsInGrid = 10,
  model_path = normalizePath("./"),
  metadata_path = normalizePath("./"),
  ModelID = "FirstModel",
  NumOfParDepPlots = 3,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  SaveInfoToPDF = TRUE,
  IfSaveModel = "mojo",
  H2OShutdown = TRUE,
  H2OStartUp = TRUE,
  DebugMode = FALSE)
```

AutoH2oMLMultiClass   *AutoH2oMLMultiClass*

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(
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
```

```
      data = NULL,
      TrainOnFull = FALSE,
      ValidationData = NULL,
      TestData = NULL,
      TargetColumnName = NULL,
      FeatureColNames = NULL,
      ExcludeAlgos = NULL,
      eval_metric = "logloss",
      MaxMem = {      gc()
       paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo",
         intern = TRUE))/1e+06)), "G") },
      NThreads = max(1, parallel::detectCores() - 2),
      MaxModelsInGrid = 2,
      model_path = NULL,
      metadata_path = NULL,
      ModelID = "FirstModel",
      ReturnModelObjects = TRUE,
      SaveModelObjects = FALSE,
      SaveInfoToPDF = TRUE,
      IfSaveModel = "mojo",
      H2OShutdown = TRUE,
      H2OStartUp = TRUE,
      DebugMode = FALSE
    )
```

Arguments

OutputSelection

You can select what type of output you want returned. Choose from c("EvalMetrics", "PDFs", "Score_TrainData")

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"

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

SaveInfoToPDF   Set to TRUE to print model insights to PDF

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

H2OStartUp   Set to FALSE

DebugMode   Set to TRUE to get a print out of steps taken internally

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(
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
  data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = "Adrian",
  FeatureColNames = names(data)[!names(data) %in% c("IDcol_1", "IDcol_2","Adrian")],
  ExcludeAlgos = NULL,
```

```
       eval_metric = "logloss",
 MaxMem = {gc();paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo", inter
 NThreads = max(1, parallel::detectCores()-2),
 MaxModelsInGrid = 10,
 model_path = normalizePath("./"),
 metadata_path = normalizePath("./"),
 ModelID = "FirstModel",
 ReturnModelObjects = TRUE,
 SaveModelObjects = FALSE,
 SaveInfoToPDF = TRUE,
 IfSaveModel = "mojo",
 H2OShutdown = TRUE,
 H2OStartUp = TRUE,
 DebugMode = FALSE)
```

AutoH2oMLRegression   *AutoH2oMLRegression*

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(
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
  data = NULL,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  ExcludeAlgos = NULL,
  TransformNumericColumns = NULL,
  Methods = c("BoxCox", "Asinh", "Log", "LogPlus1", "Sqrt", "Asin", "Logit"),
  eval_metric = "RMSE",
  MaxMem = {      gc()
   paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo",
      intern = TRUE))/1e+06)), "G") },
  NThreads = max(1, parallel::detectCores() - 2),
  model_path = NULL,
  metadata_path = NULL,
  ModelID = "FirstModel",
  NumOfParDepPlots = 3,
  ReturnModelObjects = TRUE,
```

```
      SaveModelObjects = FALSE,
      SaveInfoToPDF = TRUE,
      IfSaveModel = "mojo",
      H2OShutdown = TRUE,
      H2OStartUp = TRUE,
      DebugMode = FALSE
)
```

Arguments

OutputSelection

        You can select what type of output you want returned. Choose from c("EvalMetrics", "PDFs", "Score_TrainData")

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 "YeoJohnson", "BoxCox", "Asinh", "Log", "LogPlus1", "Sqrt", "Asin", or "Logit". If more than one is selected, the one with the best normalization pearson statistic will be used. Identity is automatically selected and compared.

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

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

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
SaveInfoToPDF	Set to TRUE to save insights to PDF
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 inside the function
H2OStartUp	Defaults to TRUE which means H2O will be started inside the function
DebugMode	Set to TRUE to print to screen steps taken internally

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(), AutoH2oGLMRegression(), AutoLightGBMRegression(), AutoNLS(), 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 = {gc();paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo", inter
  NThreads = max(1, parallel::detectCores()-2),
  H2OShutdown = TRUE,
  H2OStartUp = TRUE,
  IfSaveModel = "mojo",

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

  # Metadata arguments
  OutputSelection = c("EvalMetrics", "PDFs", "Score_TrainData"),
```

```
model_path = NULL,
metadata_path = NULL,
ModelID = "FirstModel",
ReturnModelObjects = TRUE,
SaveModelObjects = FALSE,
SaveInfoToPDF = TRUE,
DebugMode = FALSE,

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

# Model args
ExcludeAlgos = NULL)
```

AutoH2OMLScoring   *AutoH2OMLScoring*

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,
  H2OStartUp = TRUE,
  MaxMem = {      gc()
   paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo",
     intern = TRUE))/1e+06)), "G") },
  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 to shutdown H2O inside the function.
H2OStartUp	Defaults to TRUE which means H2O will be started inside the function
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(), AutoLightGBMScoring(), AutoXGBoostScoring()

Examples

```
## Not run:
Preds <- AutoH2OMLScoring(
  ScoringData = data,
  ModelObject = NULL,
  ModelType = "mojo",
  H2OShutdown = TRUE,
  H2OStartUp = TRUE,
 MaxMem = {gc();paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo", inter
  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)
```

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()](), [AutoDiffLagN()](), [AutoInteraction()](), [AutoLagRollMode()](), [AutoLagRollStatsScoring()](), [AutoLagRollStats()](), [AutoTransformationCreate()](), [AutoTransformationScore(]() [AutoWord2VecModeler()](), [AutoWord2VecScoring()](), [CategoricalEncoding()](), [CreateCalendarVariables()](), [CreateHolidayVariables()](), [DummifyDT()](), [H2OAutoencoderScoring()](), [H2OAutoencoder()](), [ModelDataPrep()](), [TimeSeriesFill()]()

AutoInteraction	*AutoInteraction*

Description

AutoInteraction creates interaction variables from your numerical features in your data. Supply a set of column names to utilize and set the interaction level. Supply a character vector of columns to exclude and the function will ignore those features.

Usage

```
AutoInteraction(
  data = NULL,
  NumericVars = NULL,
  InteractionDepth = 2,
  Center = TRUE,
  Scale = TRUE,
  SkipCols = NULL,
  Scoring = FALSE,
  File = NULL
)
```

Arguments

`data`	Source data.table
`InteractionDepth`	
	The max K in N choose K. If NULL, K will loop through 1 to length(NumVars). Default is 2 for pairwise interactions
`Center`	TRUE to center the data
`Scale`	TRUE to scale the data
`SkipCols`	Use this to exclude features from being created. An example could be, you build a model with all variables and then use the varaible importance list to determine which features aren't necessary and pass that set of features into this argument as a character vector.
`Scoring`	Defaults to FALSE. Set to TRUE for generating these columns in a model scoring setting
`File`	When Scoring is set to TRUE you have to supply either the .Rdata list with lookup values for recreating features or a pathfile to the .Rdata file with the lookup values. If you didn't center or scale the data then this argument can be ignored.
`NumVars`	Names of numeric columns (if NULL, all numeric and integer columns will be used)

Author(s)

Adrian Antico

See Also

Other Feature Engineering: `AutoDataPartition()`, `AutoDiffLagN()`, `AutoHierarchicalFourier()`, `AutoLagRollMode()`, `AutoLagRollStatsScoring()`, `AutoLagRollStats()`, `AutoTransformationCreate()`, `AutoTransformationScore()`, `AutoWord2VecModeler()`, `AutoWord2VecScoring()`, `CategoricalEncoding()`, `CreateCalendarVariables()`, `CreateHolidayVariables()`, `DummifyDT()`, `H2OAutoencoderScoring()`, `H2OAutoencoder()`, `ModelDataPrep()`, `TimeSeriesFill()`

Examples

```
## Not run:

##########################################
# Feature Engineering for Model Training
```

```
#########################################

# Create fake data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.70,
  N = 50000,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  ZIP = 0L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Print number of columns
print(ncol(data))

# Store names of numeric and integer cols
Cols <-names(data)[c(which(unlist(lapply(data, is.numeric))),
                     which(unlist(lapply(data, is.integer))))]

# Model Training Feature Engineering
system.time(data <- RemixAutoML::AutoInteraction(
  data = data,
  NumericVars = Cols,
  InteractionDepth = 4,
  Center = TRUE,
  Scale = TRUE,
  SkipCols = NULL,
  Scoring = FALSE,
  File = getwd()))

# user   system elapsed
# 0.30    0.11    0.41

# Print number of columns
print(ncol(data))

#########################################
# Feature Engineering for Model Scoring
#########################################

# Create fake data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.70,
  N = 1000,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  ZIP = 0L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Print number of columns
```

```
print(ncol(data))

# Reduce to single row to mock a scoring scenario
data <- data[1L]

# Model Scoring Feature Engineering
system.time(data <- RemixAutoML::AutoInteraction(
  data = data,
  NumericVars = names(data)[
    c(which(unlist(lapply(data, is.numeric))),
      which(unlist(lapply(data, is.integer))))],
  InteractionDepth = 4,
  Center = TRUE,
  Scale = TRUE,
  SkipCols = NULL,
  Scoring = TRUE,
  File = file.path(getwd(), "Standardize.Rdata")))

# user  system elapsed
# 0.19    0.00    0.19

# Print number of columns
print(ncol(data))

## End(Not run)
```

AutoLagRollMode   *AutoLagRollMode*

Description

Create lags and rolling modes for categorical variables.

Usage

```
AutoLagRollMode(
  data,
  Lags = 1,
  ModePeriods = 0,
  Targets = NULL,
  GroupingVars = NULL,
  SortDateName = NULL,
  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.
ModePeriods	A numberic vector of window sizes

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
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()`, `AutoDiffLagN()`, `AutoHierarchicalFourier()`, `AutoInteraction()`, `AutoLagRollStatsScoring()`, `AutoLagRollStats()`, `AutoTransformationCreate()`, `AutoTransformationScore()`, `AutoWord2VecModeler()`, `AutoWord2VecScoring()`, `CategoricalEncoding()`, `CreateCalendarVariables()`, `CreateHolidayVariables()`, `DummifyDT()`, `H2OAutoencoderScoring()`, `H2OAutoencoder()`, `ModelDataPrep()`, `TimeSeriesFill()`

Examples

```
## Not run:
# NO GROUPING CASE: Create fake Panel Data----
Count <- 1L
for(Level in LETTERS) {
  datatemp <- RemixAutoML::FakeDataGenerator(
    Correlation = 0.75,
    N = 25000L,
    ID = 0L,
    ZIP = 0L,
    FactorCount = 2L,
    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
}

# NO GROUPING CASE: Create rolling modes for categorical features
```

```
data <- RemixAutoML::AutoLagRollMode(
  data,
  Lags          = seq(1,5,1),
  ModePeriods   = seq(2,5,1),
  Targets       = c("Factor_1"),
  GroupingVars  = NULL,
  SortDateName  = "DateTime",
  WindowingLag  = 1,
  Type          = "Lag",
  SimpleImpute  = TRUE)

# GROUPING CASE: Create fake Panel Data----
Count <- 1L
for(Level in LETTERS) {
  datatemp <- RemixAutoML::FakeDataGenerator(
    Correlation = 0.75,
    N = 25000L,
    ID = 0L,
    ZIP = 0L,
    FactorCount = 2L,
    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
}

# GROUPING CASE: Create rolling modes for categorical features
data <- RemixAutoML::AutoLagRollMode(
  data,
  Lags          = seq(1,5,1),
  ModePeriods   = seq(2,5,1),
  Targets       = c("Factor_1"),
  GroupingVars  = "Factor_2",
  SortDateName  = "DateTime",
  WindowingLag  = 1,
  Type          = "Lag",
  SimpleImpute  = TRUE)

## 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 = NULL,
  TimeUnitAgg = NULL,
  TimeGroups = NULL,
  TimeBetween = NULL,
  RollOnLag1 = TRUE,
  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
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`(), `AutoDiffLagN`(), `AutoHierarchicalFourier`(), `AutoInteraction`(), `AutoLagRollMode`(), `AutoLagRollStatsScoring`(), `AutoTransformationCreate`(), `AutoTransformationScore`(), `AutoWord2VecModeler`(), `AutoWord2VecScoring`(), `CategoricalEncoding`(), `CreateCalendarVariables`(), `CreateHolidayVariables`(), `DummifyDT`(), `H2OAutoencoderScoring`(), `H2OAutoencoder`(), `ModelDataPrep`(), `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()`, `AutoDiffLagN()`, `AutoHierarchicalFourier()`, `AutoInteraction()`, `AutoLagRollMode()`, `AutoLagRollStats()`, `AutoTransformationCreate`, `AutoTransformationScore()`, `AutoWord2VecModeler()`, `AutoWord2VecScoring()`, `CategoricalEncoding()`, `CreateCalendarVariables()`, `CreateHolidayVariables()`, `DummifyDT()`, `H2OAutoencoderScoring()`, `H2OAutoencoder()`, `ModelDataPrep()`, `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)
```

AutoLightGBMCARMA   *AutoLightGBMCARMA*

Description

AutoLightGBMCARMA 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

```
AutoLightGBMCARMA(
  data = NULL,
  XREGS = NULL,
  TimeWeights = NULL,
  NonNegativePred = FALSE,
  RoundPreds = FALSE,
  TrainOnFull = FALSE,
  TargetColumnName = NULL,
  DateColumnName = NULL,
  HierarchGroups = NULL,
  GroupVariables = NULL,
  FC_Periods = 5,
  NThreads = max(1, parallel::detectCores() - 2L),
  SaveDataPath = NULL,
  PDFOutputPath = NULL,
  TimeUnit = "week",
  TimeGroups = c("weeks", "months"),
  TargetTransformation = FALSE,
  Methods = c("Asinh", "Log", "LogPlus1", "Sqrt", "Asin", "Logit"),
  EncodingMethod = "binary",
  AnomalyDetection = 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 = 0,
  CalendarVariables = c("second", "minute", "hour", "wday", "mday", "yday", "week",
      "wom", "isoweek", "month", "quarter", "year"),
   HolidayVariable = c("USPublicHolidays", "EasterGroup", "ChristmasGroup",
      "OtherEcclesticalFeasts"),
   HolidayLookback = NULL,
   HolidayLags = 1L,
   HolidayMovingAverages = 3L,
   TimeTrendVariable = FALSE,
   DataTruncate = FALSE,
   ZeroPadSeries = NULL,
   SplitRatios = c(1 - 10/100, 10/100),
   PartitionType = "random",
   Timer = TRUE,
   DebugMode = FALSE,
   GridTune = FALSE,
   GridEvalMetric = "mae",
   ModelCount = 30L,
   MaxRunsWithoutNewWinner = 20L,
   MaxRunMinutes = 24L * 60L,
   Device_Type = "cpu",
   LossFunction = "regression",
   EvalMetric = "mae",
   Input_Model = NULL,
   Task = "train",
   Boosting = "gbdt",
   LinearTree = FALSE,
   Trees = 1000,
   ETA = 0.1,
   Num_Leaves = 31,
   Deterministic = TRUE,
   Force_Col_Wise = FALSE,
   Force_Row_Wise = FALSE,
   Max_Depth = 6,
   Min_Data_In_Leaf = 20,
   Min_Sum_Hessian_In_Leaf = 0.001,
   Bagging_Freq = 1,
   Bagging_Fraction = 1,
   Feature_Fraction = 1,
   Feature_Fraction_Bynode = 1,
   Lambda_L1 = 0,
   Lambda_L2 = 0,
   Extra_Trees = FALSE,
   Early_Stopping_Round = 10,
   First_Metric_Only = TRUE,
   Max_Delta_Step = 0,
   Linear_Lambda = 0,
```

```
    Min_Gain_To_Split = 0,
    Drop_Rate_Dart = 0.1,
    Max_Drop_Dart = 50,
    Skip_Drop_Dart = 0.5,
    Uniform_Drop_Dart = FALSE,
    Top_Rate_Goss = FALSE,
    Other_Rate_Goss = FALSE,
    Monotone_Constraints = NULL,
    Monotone_Constraints_method = "advanced",
    Monotone_Penalty = 0,
    Forcedsplits_Filename = NULL,
    Refit_Decay_Rate = 0.9,
    Path_Smooth = 0,
    Max_Bin = 255,
    Min_Data_In_Bin = 3,
    Data_Random_Seed = 1,
    Is_Enable_Sparse = TRUE,
    Enable_Bundle = TRUE,
    Use_Missing = TRUE,
    Zero_As_Missing = FALSE,
    Two_Round = FALSE,
    Convert_Model = NULL,
    Convert_Model_Language = "cpp",
    Boost_From_Average = TRUE,
    Alpha = 0.9,
    Fair_C = 1,
    Poisson_Max_Delta_Step = 0.7,
    Tweedie_Variance_Power = 1.5,
    Lambdarank_Truncation_Level = 30,
    Is_Provide_Training_Metric = TRUE,
    Eval_At = c(1, 2, 3, 4, 5),
    Num_Machines = 1,
    Gpu_Platform_Id = -1,
    Gpu_Device_Id = -1,
    Gpu_Use_Dp = TRUE,
    Num_Gpu = 1
)
```

Arguments

data	Supply your full series data set here
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.
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 = 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 Group-Variables 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

NThreads   Set the maximum number of threads you'd like to dedicate to the model run. E.g. 8

SaveDataPath   Path to save modeling data

PDFOutputPath   Supply a path to save model insights to PDF

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 'YeoJohnson', 'BoxCox', 'Asinh', 'Log', 'LogPlus1', 'Sqrt', 'Asin', or 'Logit'. If more than one is selected, the one with the best normalization pearson statistic will be used. Identity is automatically selected and compared.

EncodingMethod Choose from 'binary', 'm_estimator', 'credibility', 'woe', 'target_encoding', 'poly_encode', 'backward_difference', 'helmert'

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

Lags   Select the periods for all lag variables you want to create. E.g. c(1:5,52) or list('day' = c(1:10), 'weeks' = c(1:4))

MA_Periods   Select the periods for all moving average variables you want to create. E.g. c(1:5,52) or list('day' = c(2:10), 'weeks' = c(2:4))

SD_Periods   Select the periods for all moving standard deviation variables you want to create. E.g. c(1:5,52) or list('day' = c(2:10), 'weeks' = c(2:4))

Skew_Periods   Select the periods for all moving skewness variables you want to create. E.g. c(1:5,52) or list('day' = c(2:10), 'weeks' = c(2:4))

Kurt_Periods   Select the periods for all moving kurtosis variables you want to create. E.g. c(1:5,52) or list('day' = c(2:10), 'weeks' = c(2:4))

Quantile_Periods
                Select the periods for all moving quantiles variables you want to create. E.g. c(1:5,52) or list('day' = c(2:10), 'weeks' = c(2:4))

Quantiles_Selected
                Select from the following c('q5','q10','q15','q20','q25','q30','q35','q40','q45','q50','q55','q60','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', 'wom', 'isoweek', 'month', 'quarter', 'year'

HolidayVariable
    NULL, or select from 'USPublicHolidays', 'EasterGroup', 'ChristmasGroup',
    'OtherEcclesticalFeasts'

HolidayLookback
    Number of days in range to compute number of holidays from a given date in
    the data. If NULL, the number of days are computed for you.

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   NULL to do nothing. Otherwise, set to 'maxmax', 'minmax', 'maxmin', 'min-
    min'. See TimeSeriesFill for explanations of each type

SplitRatios   E.g c(0.7,0.2,0.1) for train, validation, and test sets

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

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

MaxRunsWithoutNewWinner
    Number of consecutive runs without a new winner in order to terminate proce-
    dure

MaxRunMinutes   Default 24L*60L
    # ML Args begin

Device_Type   = 'CPU'

LossFunction   = 'regression'

EvalMetric   = 'mae'

Input_Model   = NULL

Task   = 'train'

Boosting   = 'gbdt'

LinearTree   = FALSE

Trees   = 1000

ETA   = 0.10

Num_Leaves   = 31

Deterministic   = TRUE
    # Learning Parameters # https://lightgbm.readthedocs.io/en/latest/Parameters.html#learning-
    control-parameters

```
Force_Col_Wise  = FALSE
Force_Row_Wise  = FALSE
Max_Depth       = 6
Min_Data_In_Leaf
                = 20
Min_Sum_Hessian_In_Leaf
                = 0.001
Bagging_Freq    = 1.0
Bagging_Fraction
                = 1.0
Feature_Fraction
                = 1.0
Feature_Fraction_Bynode
                = 1.0
Lambda_L1       = 0.0
Lambda_L2       = 0.0
Extra_Trees     = FALSE
Early_Stopping_Round
                = 10
First_Metric_Only
                = TRUE
Max_Delta_Step  = 0.0
Linear_Lambda   = 0.0
Min_Gain_To_Split
                = 0
Drop_Rate_Dart  = 0.10
Max_Drop_Dart   = 50
Skip_Drop_Dart  = 0.50
Uniform_Drop_Dart
                = FALSE
Top_Rate_Goss   = FALSE
Other_Rate_Goss
                = FALSE
Monotone_Constraints
                = NULL
Monotone_Constraints_method
                = 'advanced'
Monotone_Penalty
                = 0.0
Forcedsplits_Filename
                = NULL
Refit_Decay_Rate
                = 0.90
Path_Smooth     = 0.0
                # IO Dataset Parameters # https://lightgbm.readthedocs.io/en/latest/Parameters.html#io-
                parameters
Max_Bin         = 255
```

```
Min_Data_In_Bin
                = 3
Data_Random_Seed
                = 1
Is_Enable_Sparse
                = TRUE
Enable_Bundle   = TRUE
Use_Missing     = TRUE
Zero_As_Missing
                = FALSE
Two_Round       = FALSE
                # Convert Parameters
Convert_Model   = NULL
Convert_Model_Language
                = 'cpp'
                # Objective Parameters # https://lightgbm.readthedocs.io/en/latest/Parameters.html#objective-
                parameters
Boost_From_Average
                = TRUE
Alpha           = 0.90
Fair_C          = 1.0
Poisson_Max_Delta_Step
                = 0.70
Tweedie_Variance_Power
                = 1.5
Lambdarank_Truncation_Level
                = 30
                # Metric Parameters (metric is in Core) # https://lightgbm.readthedocs.io/en/latest/Parameters.html#m
                parameters
Is_Provide_Training_Metric
                = TRUE,
Eval_At         = c(1,2,3,4,5)
                # Network Parameters # https://lightgbm.readthedocs.io/en/latest/Parameters.html#network-
                parameters
Num_Machines    = 1
                # GPU Parameters
Gpu_Platform_Id
                = -1
Gpu_Device_Id   = -1
Gpu_Use_Dp      = TRUE
Num_Gpu         = 1
TreeMethod      Choose from 'hist', 'gpu_hist'
#               https://lightgbm.readthedocs.io/en/latest/Parameters.html#gpu-parameters
```

Value

See examples

Author(s)

Adrian Antico

See Also

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

Examples

```
## Not run:

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

# Ensure series have no missing dates (also remove series with more than 25% missing values)
data <- RemixAutoML::TimeSeriesFill(
  data,
  DateColumnName = 'Date',
  GroupVariables = c('Store','Dept'),
  TimeUnit = 'weeks',
  FillType = 'maxmax',
  MaxMissingPercent = 0.25,
  SimpleImpute = TRUE)

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

# Remove IsHoliday column
data[, IsHoliday := NULL]

# Create xregs (this is the include the categorical variables instead of utilizing only the interaction of them)
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))]

# Build forecast
Results <- AutoLightGBMCARMA(

  # 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
  EncodingMethod = 'binary',
  ZeroPadSeries = NULL,
```

```
DataTruncate = FALSE,
SplitRatios = c(1 - 10 / 138, 10 / 138),
PartitionType = 'timeseries',
AnomalyDetection = NULL,

# Productionize
FC_Periods = 0,
TrainOnFull = FALSE,
NThreads = 8,
Timer = TRUE,
DebugMode = FALSE,
SaveDataPath = NULL,
PDFOutputPath = NULL,

# Target Transformations
TargetTransformation = TRUE,
Methods = c('BoxCox', 'Asinh', 'Asin', 'Log',
            'LogPlus1', 'Sqrt', '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,
Quantiles_Selected = c('q5','q95'),
XREGS = xregs,
FourierTerms = 4,
CalendarVariables = c('week', 'wom', 'month', 'quarter'),
HolidayVariable = c('USPublicHolidays','EasterGroup',
  'ChristmasGroup','OtherEcclesticalFeasts'),
HolidayLookback = NULL,
HolidayLags = 1,
HolidayMovingAverages = 1:2,
TimeTrendVariable = TRUE,

# ML eval args
TreeMethod = 'hist',
EvalMetric = 'RMSE',
LossFunction = 'reg:squarederror',

# Grid tuning args
GridTune = FALSE,
GridEvalMetric = 'mae',
ModelCount = 30L,
MaxRunsWithoutNewWinner = 20L,
MaxRunMinutes = 24L*60L,

# LightGBM Args
Device_Type = TaskType,
LossFunction = 'regression',
EvalMetric = 'MAE',
Input_Model = NULL,
```

```
Task = 'train',
Boosting = 'gbdt',
LinearTree = FALSE,
Trees = 1000,
ETA = 0.10,
Num_Leaves = 31,
Deterministic = TRUE,

# Learning Parameters
# https://lightgbm.readthedocs.io/en/latest/Parameters.html#learning-control-parameters
Force_Col_Wise = FALSE,
Force_Row_Wise = FALSE,
Max_Depth = 6,
Min_Data_In_Leaf = 20,
Min_Sum_Hessian_In_Leaf = 0.001,
Bagging_Freq = 1.0,
Bagging_Fraction = 1.0,
Feature_Fraction = 1.0,
Feature_Fraction_Bynode = 1.0,
Lambda_L1 = 0.0,
Lambda_L2 = 0.0,
Extra_Trees = FALSE,
Early_Stopping_Round = 10,
First_Metric_Only = TRUE,
Max_Delta_Step = 0.0,
Linear_Lambda = 0.0,
Min_Gain_To_Split = 0,
Drop_Rate_Dart = 0.10,
Max_Drop_Dart = 50,
Skip_Drop_Dart = 0.50,
Uniform_Drop_Dart = FALSE,
Top_Rate_Goss = FALSE,
Other_Rate_Goss = FALSE,
Monotone_Constraints = NULL,
Monotone_Constraints_Method = 'advanced',
Monotone_Penalty = 0.0,
Forcedsplits_Filename = NULL, # use for AutoStack option; .json file
Refit_Decay_Rate = 0.90,
Path_Smooth = 0.0,

# IO Dataset Parameters
# https://lightgbm.readthedocs.io/en/latest/Parameters.html#io-parameters
Max_Bin = 255,
Min_Data_In_Bin = 3,
Data_Random_Seed = 1,
Is_Enable_Sparse = TRUE,
Enable_Bundle = TRUE,
Use_Missing = TRUE,
Zero_As_Missing = FALSE,
Two_Round = FALSE,

# Convert Parameters
Convert_Model = NULL,
Convert_Model_Language = 'cpp',

# Objective Parameters
# https://lightgbm.readthedocs.io/en/latest/Parameters.html#objective-parameters
```

```
      Boost_From_Average = TRUE,
      Alpha = 0.90,
      Fair_C = 1.0,
      Poisson_Max_Delta_Step = 0.70,
      Tweedie_Variance_Power = 1.5,
      Lambdarank_Truncation_Level = 30,

      # Metric Parameters (metric is in Core)
      # https://lightgbm.readthedocs.io/en/latest/Parameters.html#metric-parameters
      Is_Provide_Training_Metric = TRUE,
      Eval_At = c(1,2,3,4,5),

      # Network Parameters
      # https://lightgbm.readthedocs.io/en/latest/Parameters.html#network-parameters
      Num_Machines = 1,

      # GPU Parameters
      # https://lightgbm.readthedocs.io/en/latest/Parameters.html#gpu-parameters
      Gpu_Platform_Id = -1,
      Gpu_Device_Id = -1,
      Gpu_Use_Dp = TRUE,
      Num_Gpu = 1)

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

AutoLightGBMClassifier

AutoLightGBMClassifier

Description

AutoLightGBMClassifier is an automated lightgbm 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

```
AutoLightGBMClassifier(
  data = NULL,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
```

```
 TargetColumnName = NULL,
 FeatureColNames = NULL,
 PrimaryDateColumn = NULL,
 IDcols = NULL,
 WeightsColumnName = NULL,
 CostMatrixWeights = c(1, 0, 0, 1),
 EncodingMethod = "credibility",
OutputSelection = c("Importances", "EvalPlots", "EvalMetrics", "Score_TrainData"),
 model_path = NULL,
 metadata_path = NULL,
 DebugMode = FALSE,
 SaveInfoToPDF = FALSE,
 ModelID = "TestModel",
 ReturnFactorLevels = TRUE,
 ReturnModelObjects = TRUE,
 SaveModelObjects = FALSE,
 NumOfParDepPlots = 3L,
 Verbose = 0L,
 GridTune = FALSE,
 grid_eval_metric = "Utility",
 BaselineComparison = "default",
 MaxModelsInGrid = 10L,
 MaxRunsWithoutNewWinner = 20L,
 MaxRunMinutes = 24L * 60L,
 PassInGrid = NULL,
 input_model = NULL,
 task = "train",
 device_type = "CPU",
 NThreads = parallel::detectCores()/2,
 objective = "binary",
 metric = "binary_logloss",
 boosting = "gbdt",
 LinearTree = FALSE,
 Trees = 50L,
 eta = NULL,
 num_leaves = 31,
 deterministic = TRUE,
 force_col_wise = FALSE,
 force_row_wise = FALSE,
 max_depth = NULL,
 min_data_in_leaf = 20,
 min_sum_hessian_in_leaf = 0.001,
 bagging_freq = 0,
 bagging_fraction = 1,
 feature_fraction = 1,
 feature_fraction_bynode = 1,
 extra_trees = FALSE,
 early_stopping_round = 10,
 first_metric_only = TRUE,
 max_delta_step = 0,
 lambda_l1 = 0,
 lambda_l2 = 0,
```

```
        linear_lambda = 0,
        min_gain_to_split = 0,
        drop_rate_dart = 0.1,
        max_drop_dart = 50,
        skip_drop_dart = 0.5,
        uniform_drop_dart = FALSE,
        top_rate_goss = FALSE,
        other_rate_goss = FALSE,
        monotone_constraints = NULL,
        monotone_constraints_method = "advanced",
        monotone_penalty = 0,
        forcedsplits_filename = NULL,
        refit_decay_rate = 0.9,
        path_smooth = 0,
        max_bin = 255,
        min_data_in_bin = 3,
        data_random_seed = 1,
        is_enable_sparse = TRUE,
        enable_bundle = TRUE,
        use_missing = TRUE,
        zero_as_missing = FALSE,
        two_round = FALSE,
        convert_model = NULL,
        convert_model_language = "cpp",
        boost_from_average = TRUE,
        is_unbalance = FALSE,
        scale_pos_weight = 1,
        is_provide_training_metric = TRUE,
        eval_at = c(1, 2, 3, 4, 5),
        num_machines = 1,
        gpu_platform_id = -1,
        gpu_device_id = -1,
        gpu_use_dp = TRUE,
        num_gpu = 1
    )
```

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.
TargetColumnName	
	Either supply the target column name OR the column number where the target is located (but not mixed types).
FeatureColNames	
	Either supply the feature column names OR the column number where the target is located (but not mixed types)
PrimaryDateColumn	
	Supply a date or datetime column for catboost to utilize time as its basis for handling categorical features, instead of random shuffling

IDcols	A vector of column names or column numbers to keep in your data but not include in the modeling.
WeightsColumnName	
	Supply a column name for your weights column. Leave NULL otherwise
EncodingMethod	Choose from 'binary', 'm_estimator', 'credibility', 'woe', 'target_encoding', 'poly_encode', 'backward_difference', 'helmert'
OutputSelection	
	You can select what type of output you want returned. Choose from c("Importances", "EvalPlots", "EvalMetrics", "PDFs", "Score_TrainData")
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.
DebugMode	Set to TRUE to get a print out of the steps taken throughout the function
SaveInfoToPDF	Set to TRUE to save model insights to pdf
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
NumOfParDepPlots	
	Tell the function the number of partial dependence calibration plots you want to create.
Verbose	Set to 0 if you want to suppress model evaluation updates in training
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	
	"mae", "mape", "rmse", "r2". Case sensitive
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.
	# Core parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#core-parameter
MaxModelsInGrid	
	Number of models to test from grid options (243 total possible options)
MaxRunsWithoutNewWinner	
	Runs without new winner to end procedure
MaxRunMinutes	In minutes
PassInGrid	Default is NULL. Provide a data.table of grid options from a previous run.
input_model	= NULL, # continue training a model that is stored to fil
task	'train' or 'refit'
device_type	'cpu' or 'gpu'
NThreads	only list up to number of cores, not threads. parallel::detectCores() / 2
objective	'binary'

metric	'binary_logloss', 'average_precision', 'auc', 'map', 'binary_error', 'auc_mu'
boosting	'gbdt', 'rf', 'dart', 'goss'
LinearTree	FALSE
Trees	50L
eta	NULL
num_leaves	31
deterministic	TRUE
	# Learning Parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#learning-control-parameter
force_col_wise	FALSE
force_row_wise	FALSE
max_depth	NULL
min_data_in_leaf	
	20
min_sum_hessian_in_leaf	
	0.001
bagging_freq	0
bagging_fraction	
	1.0
feature_fraction	
	1.0
feature_fraction_bynode	
	1.0
extra_trees	FALSE
early_stopping_round	
	10
first_metric_only	
	TRUE
max_delta_step	0.0
lambda_l1	0.0
lambda_l2	0.0
linear_lambda	0.0
min_gain_to_split	
	0
drop_rate_dart	0.10
max_drop_dart	50
skip_drop_dart	0.50
uniform_drop_dart	
	FALSE
top_rate_goss	FALSE
other_rate_goss	
	FALSE
monotone_constraints	
	"gbdt_prediction.cpp"
monotone_constraints_method	
	'advanced'

```
monotone_penalty
                0.0
forcedsplits_filename
                NULL # use for AutoStack option; .json fil
refit_decay_rate
                0.90
path_smooth     0.0
                # IO Dataset Parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#io-
                parameters
max_bin         255
min_data_in_bin
                3
data_random_seed
                1
is_enable_sparse
                TRUE
enable_bundle   TRUE
use_missing     TRUE
zero_as_missing
                FALSE
two_round       FALSE
                # Convert Parameters # https://lightgbm.readthedocs.io/en/latest/Parameters.html#convert-
                parameters
convert_model   'gbdt_prediction.cpp'
convert_model_language
                'cpp'
                # Objective Parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#objective-
                parameters
boost_from_average
                TRUE
is_unbalance    FALSE
scale_pos_weight
                1.0
                # Metric Parameters (metric is in Core)
is_provide_training_metric
                TRUE
eval_at         c(1,2,3,4,5)
                # Network Parameter
num_machines    1
                # GPU Parameter
gpu_platform_id
                -1
gpu_device_id   -1
gpu_use_dp      TRUE
num_gpu         1
```

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, and GridList

Author(s)

Adrian Antico

See Also

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

Examples

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

# Run function
TestModel <- RemixAutoML::AutoLightGBMClassifier(

  # Metadata args
  OutputSelection = c('Importances','EvalPlots','EvalMetrics','Score_TrainData'),
  model_path = normalizePath("./"),
  metadata_path = NULL,
  ModelID = "Test_Model_1",
  NumOfParDepPlots = 3L,
  EncodingMethod = "credibility",
  ReturnFactorLevels = TRUE,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  SaveInfoToPDF = FALSE,
  DebugMode = FALSE,

  # Data args
  data = data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = "Adrian",
  FeatureColNames = names(data)[!names(data) %in% c("IDcol_1", "IDcol_2","Adrian")],
  PrimaryDateColumn = NULL,
  WeightsColumnName = NULL,
  IDcols = c("IDcol_1","IDcol_2"),

  # Grid parameters
```

```
GridTune = FALSE,
grid_eval_metric = 'Utility',
BaselineComparison = 'default',
MaxModelsInGrid = 10L,
MaxRunsWithoutNewWinner = 20L,
MaxRunMinutes = 24L*60L,
PassInGrid = NULL,

# Core parameters
# https://lightgbm.readthedocs.io/en/latest/Parameters.html#core-parameters
input_model = NULL, # continue training a model that is stored to file
task = "train",
device_type = 'CPU',
NThreads = parallel::detectCores() / 2,
objective = 'binary',
metric = 'binary_logloss',
boosting = 'gbdt',
LinearTree = FALSE,
Trees = 50L,
eta = NULL,
num_leaves = 31,
deterministic = TRUE,

# Learning Parameters
# https://lightgbm.readthedocs.io/en/latest/Parameters.html#learning-control-parameters
force_col_wise = FALSE,
force_row_wise = FALSE,
max_depth = NULL,
min_data_in_leaf = 20,
min_sum_hessian_in_leaf = 0.001,
bagging_freq = 0,
bagging_fraction = 1.0,
feature_fraction = 1.0,
feature_fraction_bynode = 1.0,
extra_trees = FALSE,
early_stopping_round = 10,
first_metric_only = TRUE,
max_delta_step = 0.0,
lambda_l1 = 0.0,
lambda_l2 = 0.0,
linear_lambda = 0.0,
min_gain_to_split = 0,
drop_rate_dart = 0.10,
max_drop_dart = 50,
skip_drop_dart = 0.50,
uniform_drop_dart = FALSE,
top_rate_goss = FALSE,
other_rate_goss = FALSE,
monotone_constraints = NULL,
monotone_constraints_method = "advanced",
monotone_penalty = 0.0,
forcedsplits_filename = NULL, # use for AutoStack option; .json file
refit_decay_rate = 0.90,
path_smooth = 0.0,

# IO Dataset Parameters
# https://lightgbm.readthedocs.io/en/latest/Parameters.html#io-parameters
```

```
  max_bin = 255,
  min_data_in_bin = 3,
  data_random_seed = 1,
  is_enable_sparse = TRUE,
  enable_bundle = TRUE,
  use_missing = TRUE,
  zero_as_missing = FALSE,
  two_round = FALSE,

  # Convert Parameters
  convert_model = NULL,
  convert_model_language = "cpp",

  # Objective Parameters
  # https://lightgbm.readthedocs.io/en/latest/Parameters.html#objective-parameters
  boost_from_average = TRUE,
  is_unbalance = FALSE,
  scale_pos_weight = 1.0,

  # Metric Parameters (metric is in Core)
  # https://lightgbm.readthedocs.io/en/latest/Parameters.html#metric-parameters
  is_provide_training_metric = TRUE,
  eval_at = c(1,2,3,4,5),

  # Network Parameters
  # https://lightgbm.readthedocs.io/en/latest/Parameters.html#network-parameters
  num_machines = 1,

  # GPU Parameters
  # https://lightgbm.readthedocs.io/en/latest/Parameters.html#gpu-parameters
  gpu_platform_id = -1,
  gpu_device_id = -1,
  gpu_use_dp = TRUE,
  num_gpu = 1)

## End(Not run)
```

AutoLightGBMFunnelCARMA

AutoLightGBMFunnelCARMA

Description

AutoLightGBMFunnelCARMA is a forecasting model for cohort funnel forecasting for grouped data or non-grouped data

Usage

```
AutoLightGBMFunnelCARMA(
  data,
  GroupVariables = NULL,
  BaseFunnelMeasure = NULL,
  ConversionMeasure = NULL,
  ConversionRateMeasure = NULL,
```

```
 CohortPeriodsVariable = NULL,
 CalendarDate = NULL,
 CohortDate = NULL,
 EncodingMethod = "credibility",
 OutputSelection = c("Importances", "EvalPlots", "EvalMetrics", "Score_TrainData"),
 WeightsColumnName = NULL,
 TruncateDate = NULL,
 PartitionRatios = c(0.7, 0.2, 0.1),
 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,
 DebugMode = FALSE,
 CalendarVariables = c("wday", "mday", "yday", "week", "isoweek", "month", "quarter",
   "year"),
 HolidayGroups = c("USPublicHolidays", "EasterGroup", "ChristmasGroup",
   "OtherEcclesticalFeasts"),
 HolidayLookback = NULL,
 CohortHolidayLags = c(1L, 2L, 7L),
 CohortHolidayMovingAverages = c(3L, 7L),
 CalendarHolidayLags = c(1L, 2L, 7L),
 CalendarHolidayMovingAverages = c(3L, 7L),
 ImputeRollStats = -0.001,
 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,
LossFunction = "regression",
EvalMetric = "mae",
GridEvalMetric = "mae",
NumOfParDepPlots = 1L,
Device_Type = "CPU",
Input_Model = NULL,
Task = "train",
Boosting = "gbdt",
LinearTree = FALSE,
Trees = 1000,
ETA = 0.1,
Num_Leaves = 31,
Deterministic = TRUE,
NThreads = parallel::detectCores()/2,
SaveInfoToPDF = FALSE,
Force_Col_Wise = FALSE,
Force_Row_Wise = FALSE,
Max_Depth = 6,
Min_Data_In_Leaf = 20,
Min_Sum_Hessian_In_Leaf = 0.001,
Bagging_Freq = 1,
Bagging_Fraction = 1,
Feature_Fraction = 1,
Feature_Fraction_Bynode = 1,
Lambda_L1 = 0,
Lambda_L2 = 0,
Extra_Trees = FALSE,
Early_Stopping_Round = 10,
First_Metric_Only = TRUE,
Max_Delta_Step = 0,
Linear_Lambda = 0,
Min_Gain_To_Split = 0,
Drop_Rate_Dart = 0.1,
Max_Drop_Dart = 50,
Skip_Drop_Dart = 0.5,
Uniform_Drop_Dart = FALSE,
Top_Rate_Goss = FALSE,
Other_Rate_Goss = FALSE,
Monotone_Constraints = NULL,
Monotone_Constraints_method = "advanced",
Monotone_Penalty = 0,
Forcedsplits_Filename = NULL,
Refit_Decay_Rate = 0.9,
Path_Smooth = 0,
Max_Bin = 255,
Min_Data_In_Bin = 3,
Data_Random_Seed = 1,
Is_Enable_Sparse = TRUE,
Enable_Bundle = TRUE,
Use_Missing = TRUE,
```

```
    Zero_As_Missing = FALSE,
    Two_Round = FALSE,
    Convert_Model = NULL,
    Convert_Model_Language = "cpp",
    Boost_From_Average = TRUE,
    Alpha = 0.9,
    Fair_C = 1,
    Poisson_Max_Delta_Step = 0.7,
    Tweedie_Variance_Power = 1.5,
    Lambdarank_Truncation_Level = 30,
    Is_Provide_Training_Metric = TRUE,
    Eval_At = c(1, 2, 3, 4, 5),
    Num_Machines = 1,
    Gpu_Platform_Id = -1,
    Gpu_Device_Id = -1,
    Gpu_Use_Dp = TRUE,
    Num_Gpu = 1
)
```

Arguments

data	data object
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
OutputSelection	
	= c('Importances', 'EvalPlots', 'EvalMetrics', 'Score_TrainData')
WeightsColumnName	
	= NULL
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.
PartitionRatios	
	Requires three values for train, validation, and test data sets
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.

DebugMode   Internal use

CalendarVariables
> "wday", "mday", "yday", "week", "isoweek", "month", "quarter", "year"

HolidayGroups   c("USPublicHolidays","EasterGroup","ChristmasGroup","OtherEcclesticalFeasts")

HolidayLookback
> Number of days in range to compute number of holidays from a given date in the data. If NULL, the number of days are computed for you.

CohortHolidayLags
> c(1L, 2L, 7L),

CohortHolidayMovingAverages
> c(3L, 7L),

CalendarHolidayLags
> c(1L, 2L, 7L),

CalendarHolidayMovingAverages
> = c(3L, 7L),

ImputeRollStats
> Constant value to fill NA after running AutoLagRollStats()

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"

        # Grid tuning

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

        # ML Args begin

LossFunction = 'regression'

EvalMetric   = 'mae'

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)

Device_Type   = 'CPU'

Input_Model   = NULL

Task   = 'train'

```
Boosting        = 'gbdt'
LinearTree      = FALSE
Trees           = 1000
ETA             = 0.10
Num_Leaves      = 31
Deterministic   = TRUE
                # Learning Parameters # https://lightgbm.readthedocs.io/en/latest/Parameters.html#learning-
                control-parameters
NThreads        = parallel::detectCores() / 2
Force_Col_Wise  = FALSE
Force_Row_Wise  = FALSE
Max_Depth       = 6
Min_Data_In_Leaf
                = 20
Min_Sum_Hessian_In_Leaf
                = 0.001
Bagging_Freq    = 1.0
Bagging_Fraction
                = 1.0
Feature_Fraction
                = 1.0
Feature_Fraction_Bynode
                = 1.0
Lambda_L1       = 0.0
Lambda_L2       = 0.0
Extra_Trees     = FALSE
Early_Stopping_Round
                = 10
First_Metric_Only
                = TRUE
Max_Delta_Step  = 0.0
Linear_Lambda   = 0.0
Min_Gain_To_Split
                = 0
Drop_Rate_Dart  = 0.10
Max_Drop_Dart   = 50
Skip_Drop_Dart  = 0.50
Uniform_Drop_Dart
                = FALSE
Top_Rate_Goss   = FALSE
Other_Rate_Goss
                = FALSE
Monotone_Constraints
                = NULL
Monotone_Constraints_method
                = 'advanced'
```

```
Monotone_Penalty
                = 0.0
Forcedsplits_Filename
                = NULL
Refit_Decay_Rate
                = 0.90
Path_Smooth     = 0.0
```
                # IO Dataset Parameters # https://lightgbm.readthedocs.io/en/latest/Parameters.html#io-parameters
```
Max_Bin         = 255
Min_Data_In_Bin
                = 3
Data_Random_Seed
                = 1
Is_Enable_Sparse
                = TRUE
Enable_Bundle   = TRUE
Use_Missing     = TRUE
Zero_As_Missing
                = FALSE
Two_Round       = FALSE
```
                # Convert Parameters
```
Convert_Model   = NULL
Convert_Model_Language
                = 'cpp'
```
                # Objective Parameters # https://lightgbm.readthedocs.io/en/latest/Parameters.html#objective-parameters
```
Boost_From_Average
                = TRUE
Alpha           = 0.90
Fair_C          = 1.0
Poisson_Max_Delta_Step
                = 0.70
Tweedie_Variance_Power
                = 1.5
Lambdarank_Truncation_Level
                = 30
```
                # Metric Parameters (metric is in Core) # https://lightgbm.readthedocs.io/en/latest/Parameters.html#metric-parameters
```
Is_Provide_Training_Metric
                = TRUE,
Eval_At         = c(1,2,3,4,5)
```
                # Network Parameters # https://lightgbm.readthedocs.io/en/latest/Parameters.html#network-parameters
```
Num_Machines    = 1
```
                # GPU Parameters
```
Gpu_Platform_Id
                = -1
```

```
Gpu_Device_Id    = -1
Gpu_Use_Dp       = TRUE
Num_Gpu          = 1
#                  https://lightgbm.readthedocs.io/en/latest/Parameters.html#gpu-parameters
```

Author(s)

Adrian Antico

See Also

Other Automated Funnel Data Forecasting: [AutoCatBoostFunnelCARMAScoring](), [AutoCatBoostFunnelCARMA](), [AutoLightGBMFunnelCARMAScoring](), [AutoXGBoostFunnelCARMAScoring](), [AutoXGBoostFunnelCARMA]()

Examples

```
## Not run:
# Create Fake Data
data <- RemixAutoML::FakeDataGenerator(ChainLadderData = TRUE)

# Subset data for training
ModelDataBase <- data[CalendarDateColumn < '2020-01-01' & CohortDateColumn < '2020-01-01']
ModelData <- data.table::copy(ModelDataBase)

# Train Funne Model
TestModel <- RemixAutoML::AutoLightGBMFunnelCARMA(

  # Data Arguments
  data = ModelData,
  GroupVariables = NULL,
 BaseFunnelMeasure = "Leads", # if you have XREGS, supply vector such as c("Leads", "XREGS1", "XREGS2")
  ConversionMeasure = "Appointments",
  ConversionRateMeasure = NULL,
  CohortPeriodsVariable = "CohortDays",
  WeightsColumnName = NULL,
  CalendarDate = "CalendarDateColumn",
  CohortDate = "CohortDateColumn",
  PartitionRatios = c(0.70,0.20,0.10),
  TruncateDate = NULL,
  TimeUnit = "days",
  TransformTargetVariable = TRUE,
  TransformMethods = c("Asinh","Asin","Log","LogPlus1","Sqrt","Logit"),
  AnomalyDetection = list(tstat_high = 3, tstat_low = -2),

  # MetaData Arguments
  Jobs = c("eval","train"),
  SaveModelObjects = FALSE,
  ModelID = "ModelTest",
  ModelPath = getwd(),
  MetaDataPath = NULL,
  DebugMode = TRUE,
  NumOfParDepPlots = 1L,
  EncodingMethod = "credibility",
  NThreads = parallel::detectCores(),

  # Feature Engineering Arguments
```

```
 CalendarTimeGroups = c("days","weeks","months"),
 CohortTimeGroups = c("days", "weeks"),
 CalendarVariables = c("wday","mday","yday","week","month","quarter","year"),
HolidayGroups = c("USPublicHolidays","EasterGroup","ChristmasGroup","OtherEcclesticalFeasts"),
 HolidayLookback = NULL,
 CohortHolidayLags = c(1L,2L,7L),
 CohortHolidayMovingAverages = c(3L,7L),
 CalendarHolidayLags = c(1L,2L,7L),
 CalendarHolidayMovingAverages = c(3L,7L),

 # Time Series Features
 ImputeRollStats = -0.001,
 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,12
 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",

 # ML Grid Tuning
 PassInGrid = NULL,
 GridTune = FALSE,
 BaselineComparison = "default",
 MaxModelsInGrid = 25L,
 MaxRunMinutes = 180L,
 MaxRunsWithoutNewWinner = 10L,

 # ML Setup Parameters
 LossFunction = 'regression',
 EvalMetric = 'mae',
 GridEvalMetric = 'mae',

 # LightGBM Args
 Device_Type = 'CPU',
 Input_Model = NULL,
 Task = 'train',
 Boosting = 'gbdt',
 LinearTree = FALSE,
 Trees = 50,
 ETA = 0.10,
 Num_Leaves = 31,
 Deterministic = TRUE,

 # Learning Parameters
 # https://lightgbm.readthedocs.io/en/latest/Parameters.html#learning-control-parameters
 Force_Col_Wise = FALSE,
 Force_Row_Wise = FALSE,
 Max_Depth = 6,
 Min_Data_In_Leaf = 20,
```

```
Min_Sum_Hessian_In_Leaf = 0.001,
Bagging_Freq = 1.0,
Bagging_Fraction = 1.0,
Feature_Fraction = 1.0,
Feature_Fraction_Bynode = 1.0,
Lambda_L1 = 0.0,
Lambda_L2 = 0.0,
Extra_Trees = FALSE,
Early_Stopping_Round = 10,
First_Metric_Only = TRUE,
Max_Delta_Step = 0.0,
Linear_Lambda = 0.0,
Min_Gain_To_Split = 0,
Drop_Rate_Dart = 0.10,
Max_Drop_Dart = 50,
Skip_Drop_Dart = 0.50,
Uniform_Drop_Dart = FALSE,
Top_Rate_Goss = FALSE,
Other_Rate_Goss = FALSE,
Monotone_Constraints = NULL,
Monotone_Constraints_method = 'advanced',
Monotone_Penalty = 0.0,
Forcedsplits_Filename = NULL, # use for AutoStack option; .json file
Refit_Decay_Rate = 0.90,
Path_Smooth = 0.0,

# IO Dataset Parameters
# https://lightgbm.readthedocs.io/en/latest/Parameters.html#io-parameters
Max_Bin = 255,
Min_Data_In_Bin = 3,
Data_Random_Seed = 1,
Is_Enable_Sparse = TRUE,
Enable_Bundle = TRUE,
Use_Missing = TRUE,
Zero_As_Missing = FALSE,
Two_Round = FALSE,

# Convert Parameters
Convert_Model = NULL,
Convert_Model_Language = 'cpp',

# Objective Parameters
# https://lightgbm.readthedocs.io/en/latest/Parameters.html#objective-parameters
Boost_From_Average = TRUE,
Alpha = 0.90,
Fair_C = 1.0,
Poisson_Max_Delta_Step = 0.70,
Tweedie_Variance_Power = 1.5,
Lambdarank_Truncation_Level = 30,

# Metric Parameters (metric is in Core)
# https://lightgbm.readthedocs.io/en/latest/Parameters.html#metric-parameters
Is_Provide_Training_Metric = TRUE,
Eval_At = c(1,2,3,4,5),

# Network Parameters
# https://lightgbm.readthedocs.io/en/latest/Parameters.html#network-parameters
```

```
    Num_Machines = 1,

    # GPU Parameters
    # https://lightgbm.readthedocs.io/en/latest/Parameters.html#gpu-parameters
    Gpu_Platform_Id = -1,
    Gpu_Device_Id = -1,
    Gpu_Use_Dp = TRUE,
    Num_Gpu = 1)

# Separate out the Base Funnel Measures Data
LeadsData <- data[, lapply(.SD, data.table::first), .SDcols = c("Leads"), by = c("CalendarDateColumn")]
ModelData <- ModelDataBase[, Leads := NULL]

# Forecast Funnel Model
Test <- RemixAutoML::AutoLightGBMFunnelCARMAScoring(
  TrainData = ModelData,
  ForwardLookingData = LeadsData,
  TrainEndDate = ModelData[, max(CalendarDateColumn)],
  ForecastEndDate = LeadsData[, max(CalendarDateColumn)],
  TrainOutput = TestModel$ModelOutput,
  ArgsList = TestModel$ArgsList,
  ModelPath = NULL,
  MaxCohortPeriod = 15,
  DebugMode = TRUE)

## End(Not run)
```

AutoLightGBMFunnelCARMAScoring

AutoLightGBMFunnelCARMAScoring

Description

AutoLightGBMFunnelCARMAScoring for generating forecasts

Usage

```
AutoLightGBMFunnelCARMAScoring(
  TrainData,
  ForwardLookingData = NULL,
  TrainEndDate = NULL,
  ForecastEndDate = NULL,
  ArgsList = NULL,
  TrainOutput = NULL,
  ModelPath = NULL,
  MaxCohortPeriod = NULL,
  DebugMode = FALSE
)
```

Arguments

TrainData   Data utilized in training. Do not put the BaseFunnelMeasure in this data set. Put
               it in the ForwardLookingData object

ForwardLookingData

          Base funnel measure data. Needs to cover the span of the forecast horizon

TrainEndDate   Max date from the training data

ForecastEndDate

          Max date to forecast out to

ArgsList   Output list from AutoCatBoostFunnelCARMA

TrainOutput   Pass in the model object to speed up forecasting

ModelPath   Path to model location

MaxCohortPeriod

          Max cohort periods to utilize when forecasting

DebugMode   For debugging issues

Author(s)

Adrian Antico

See Also

Other Automated Funnel Data Forecasting: `AutoCatBoostFunnelCARMAScoring()`, `AutoCatBoostFunnelCARMA()`, `AutoLightGBMFunnelCARMA()`, `AutoXGBoostFunnelCARMAScoring()`, `AutoXGBoostFunnelCARMA()`

Examples

```
## Not run:
# Create Fake Data
data <- RemixAutoML::FakeDataGenerator(ChainLadderData = TRUE)

# Subset data for training
ModelDataBase <- data[CalendarDateColumn < '2020-01-01' & CohortDateColumn < '2020-01-01']
ModelData <- data.table::copy(ModelDataBase)

# Train Funne Model
TestModel <- RemixAutoML::AutoLightGBMFunnelCARMA(

  # Data Arguments
  data = ModelData,
  GroupVariables = NULL,
 BaseFunnelMeasure = "Leads", # if you have XREGS, supply vector such as c("Leads", "XREGS1", "XREGS2")
  ConversionMeasure = "Appointments",
  ConversionRateMeasure = NULL,
  CohortPeriodsVariable = "CohortDays",
  WeightsColumnName = NULL,
  CalendarDate = "CalendarDateColumn",
  CohortDate = "CohortDateColumn",
  PartitionRatios = c(0.70,0.20,0.10),
  TruncateDate = NULL,
  TimeUnit = "days",
  TransformTargetVariable = TRUE,
  TransformMethods = c("Asinh","Asin","Log","LogPlus1","Sqrt","Logit"),
  AnomalyDetection = list(tstat_high = 3, tstat_low = -2),

  # MetaData Arguments
  Jobs = c("eval","train"),
  SaveModelObjects = FALSE,
```

```
ModelID = "ModelTest",
ModelPath = getwd(),
MetaDataPath = NULL,
DebugMode = TRUE,
NumOfParDepPlots = 1L,
EncodingMethod = "credibility",
NThreads = parallel::detectCores(),

# Feature Engineering Arguments
CalendarTimeGroups = c("days","weeks","months"),
CohortTimeGroups = c("days", "weeks"),
CalendarVariables = c("wday","mday","yday","week","month","quarter","year"),
HolidayGroups = c("USPublicHolidays","EasterGroup","ChristmasGroup","OtherEcclesticalFeasts"),
HolidayLookback = NULL,
CohortHolidayLags = c(1L,2L,7L),
CohortHolidayMovingAverages = c(3L,7L),
CalendarHolidayLags = c(1L,2L,7L),
CalendarHolidayMovingAverages = c(3L,7L),

# Time Series Features
ImputeRollStats = -0.001,
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,12
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",

# ML Grid Tuning
PassInGrid = NULL,
GridTune = FALSE,
BaselineComparison = "default",
MaxModelsInGrid = 25L,
MaxRunMinutes = 180L,
MaxRunsWithoutNewWinner = 10L,

# ML Setup Parameters
LossFunction = 'regression',
EvalMetric = 'mae',
GridEvalMetric = 'mae',

# LightGBM Args
Device_Type = 'CPU',
Input_Model = NULL,
Task = 'train',
Boosting = 'gbdt',
LinearTree = FALSE,
Trees = 50,
ETA = 0.10,
```

```
 Num_Leaves = 31,
 Deterministic = TRUE,

 # Learning Parameters
 # https://lightgbm.readthedocs.io/en/latest/Parameters.html#learning-control-parameters
 Force_Col_Wise = FALSE,
 Force_Row_Wise = FALSE,
 Max_Depth = 6,
 Min_Data_In_Leaf = 20,
 Min_Sum_Hessian_In_Leaf = 0.001,
 Bagging_Freq = 1.0,
 Bagging_Fraction = 1.0,
 Feature_Fraction = 1.0,
 Feature_Fraction_Bynode = 1.0,
 Lambda_L1 = 0.0,
 Lambda_L2 = 0.0,
 Extra_Trees = FALSE,
 Early_Stopping_Round = 10,
 First_Metric_Only = TRUE,
 Max_Delta_Step = 0.0,
 Linear_Lambda = 0.0,
 Min_Gain_To_Split = 0,
 Drop_Rate_Dart = 0.10,
 Max_Drop_Dart = 50,
 Skip_Drop_Dart = 0.50,
 Uniform_Drop_Dart = FALSE,
 Top_Rate_Goss = FALSE,
 Other_Rate_Goss = FALSE,
 Monotone_Constraints = NULL,
 Monotone_Constraints_method = 'advanced',
 Monotone_Penalty = 0.0,
 Forcedsplits_Filename = NULL, # use for AutoStack option; .json file
 Refit_Decay_Rate = 0.90,
 Path_Smooth = 0.0,

 # IO Dataset Parameters
 # https://lightgbm.readthedocs.io/en/latest/Parameters.html#io-parameters
 Max_Bin = 255,
 Min_Data_In_Bin = 3,
 Data_Random_Seed = 1,
 Is_Enable_Sparse = TRUE,
 Enable_Bundle = TRUE,
 Use_Missing = TRUE,
 Zero_As_Missing = FALSE,
 Two_Round = FALSE,

 # Convert Parameters
 Convert_Model = NULL,
 Convert_Model_Language = 'cpp',

 # Objective Parameters
 # https://lightgbm.readthedocs.io/en/latest/Parameters.html#objective-parameters
 Boost_From_Average = TRUE,
 Alpha = 0.90,
 Fair_C = 1.0,
 Poisson_Max_Delta_Step = 0.70,
 Tweedie_Variance_Power = 1.5,
```

```
    Lambdarank_Truncation_Level = 30,

    # Metric Parameters (metric is in Core)
    # https://lightgbm.readthedocs.io/en/latest/Parameters.html#metric-parameters
    Is_Provide_Training_Metric = TRUE,
    Eval_At = c(1,2,3,4,5),

    # Network Parameters
    # https://lightgbm.readthedocs.io/en/latest/Parameters.html#network-parameters
    Num_Machines = 1,

    # GPU Parameters
    # https://lightgbm.readthedocs.io/en/latest/Parameters.html#gpu-parameters
    Gpu_Platform_Id = -1,
    Gpu_Device_Id = -1,
    Gpu_Use_Dp = TRUE,
    Num_Gpu = 1)

# Separate out the Base Funnel Measures Data
LeadsData <- data[, lapply(.SD, data.table::first), .SDcols = c("Leads"), by = c("CalendarDateColumn")]
ModelData <- ModelDataBase[, Leads := NULL]

# Forecast Funnel Model
Test <- RemixAutoML::AutoLightGBMFunnelCARMAScoring(
  TrainData = ModelData,
  ForwardLookingData = LeadsData,
  TrainEndDate = ModelData[, max(CalendarDateColumn)],
  ForecastEndDate = LeadsData[, max(CalendarDateColumn)],
  TrainOutput = TestModel$ModelOutput,
  ArgsList = TestModel$ArgsList,
  ModelPath = NULL,
  MaxCohortPeriod = 15,
  DebugMode = TRUE)

## End(Not run)
```

AutoLightGBMHurdleCARMA

AutoLightGBMHurdleCARMA

Description

AutoLightGBMHurdleCARMA 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

```
AutoLightGBMHurdleCARMA(
  data,
```

```
NonNegativePred = FALSE,
Threshold = NULL,
RoundPreds = FALSE,
TrainOnFull = FALSE,
TargetColumnName = "Target",
DateColumnName = "DateTime",
HierarchGroups = NULL,
GroupVariables = NULL,
EncodingMethod = "credibility",
TimeWeights = 1,
FC_Periods = 30,
TimeUnit = "week",
TimeGroups = c("weeks", "months"),
NumOfParDepPlots = 10L,
TargetTransformation = FALSE,
Methods = c("BoxCox", "Asinh", "Log", "LogPlus1", "Sqrt", "Asin", "Logit"),
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",
    "wom", "isoweek", "month", "quarter", "year"),
HolidayVariable = c("USPublicHolidays", "EasterGroup", "ChristmasGroup",
    "OtherEcclesticalFeasts"),
HolidayLookback = NULL,
HolidayLags = 1L,
HolidayMovingAverages = 1L:2L,
TimeTrendVariable = FALSE,
ZeroPadSeries = NULL,
DataTruncate = FALSE,
SplitRatios = c(0.7, 0.2, 0.1),
PartitionType = "timeseries",
Timer = TRUE,
DebugMode = FALSE,
EvalMetric = "RMSE",
GridTune = FALSE,
PassInGrid = NULL,
ModelCount = 100,
MaxRunsWithoutNewWinner = 50,
MaxRunMinutes = 24L * 60L,
input_model = list(classifier = NULL, regression = NULL),
task = list(classifier = "train", regression = "train"),
device_type = list(classifier = "CPU", regression = "CPU"),
objective = list(classifier = "binary", regression = "regression"),
metric = list(classifier = "binary_logloss", regression = "rmse"),
```

```
boosting = list(classifier = "gbdt", regression = "gbdt"),
LinearTree = list(classifier = FALSE, regression = FALSE),
Trees = list(classifier = 1000L, regression = 1000L),
eta = list(classifier = NULL, regression = NULL),
num_leaves = list(classifier = 31, regression = 31),
deterministic = list(classifier = TRUE, regression = TRUE),
force_col_wise = list(classifier = FALSE, regression = FALSE),
force_row_wise = list(classifier = FALSE, regression = FALSE),
max_depth = list(classifier = NULL, regression = NULL),
min_data_in_leaf = list(classifier = 20, regression = 20),
min_sum_hessian_in_leaf = list(classifier = 0.001, regression = 0.001),
bagging_freq = list(classifier = 0, regression = 0),
bagging_fraction = list(classifier = 1, regression = 1),
feature_fraction = list(classifier = 1, regression = 1),
feature_fraction_bynode = list(classifier = 1, regression = 1),
extra_trees = list(classifier = FALSE, regression = FALSE),
early_stopping_round = list(classifier = 10, regression = 10),
first_metric_only = list(classifier = TRUE, regression = TRUE),
max_delta_step = list(classifier = 0, regression = 0),
lambda_l1 = list(classifier = 0, regression = 0),
lambda_l2 = list(classifier = 0, regression = 0),
linear_lambda = list(classifier = 0, regression = 0),
min_gain_to_split = list(classifier = 0, regression = 0),
drop_rate_dart = list(classifier = 0.1, regression = 0.1),
max_drop_dart = list(classifier = 50, regression = 50),
skip_drop_dart = list(classifier = 0.5, regression = 0.5),
uniform_drop_dart = list(classifier = FALSE, regression = FALSE),
top_rate_goss = list(classifier = FALSE, regression = FALSE),
other_rate_goss = list(classifier = FALSE, regression = FALSE),
monotone_constraints = list(classifier = NULL, regression = NULL),
monotone_constraints_method = list(classifier = "advanced", regression = "advanced"),
monotone_penalty = list(classifier = 0, regression = 0),
forcedsplits_filename = list(classifier = NULL, regression = NULL),
refit_decay_rate = list(classifier = 0.9, regression = 0.9),
path_smooth = list(classifier = 0, regression = 0),
max_bin = list(classifier = 255, regression = 255),
min_data_in_bin = list(classifier = 3, regression = 3),
data_random_seed = list(classifier = 1, regression = 1),
is_enable_sparse = list(classifier = TRUE, regression = TRUE),
enable_bundle = list(classifier = TRUE, regression = TRUE),
use_missing = list(classifier = TRUE, regression = TRUE),
zero_as_missing = list(classifier = FALSE, regression = FALSE),
two_round = list(classifier = FALSE, regression = FALSE),
convert_model = list(classifier = NULL, regression = NULL),
convert_model_language = list(classifier = "cpp", regression = "cpp"),
boost_from_average = list(classifier = TRUE, regression = TRUE),
is_unbalance = list(classifier = FALSE, regression = FALSE),
scale_pos_weight = list(classifier = 1, regression = 1),
is_provide_training_metric = list(classifier = TRUE, regression = TRUE),
eval_at = list(classifier = c(1, 2, 3, 4, 5), regression = c(1, 2, 3, 4, 5)),
num_machines = list(classifier = 1, regression = 1),
gpu_platform_id = list(classifier = -1, regression = -1),
```

```
    gpu_device_id = list(classifier = -1, regression = -1),
    gpu_use_dp = list(classifier = TRUE, regression = TRUE),
    num_gpu = list(classifier = 1, regression = 1)
)
```

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.
`EncodingMethod`	Choose from 'binary', 'poly_encode', 'backward_difference', 'helmert' for multiclass cases and additionally 'm_estimator', 'credibility', 'woe', 'target_encoding' for classification use cases.
`TimeWeights`	Timeweights creation. Supply a value, such as 0.9999
`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`	Choose from 'YeoJohnson', 'BoxCox', 'Asinh', 'Log', 'LogPlus1', 'Sqrt', 'Asin', or 'Logit'. If more than one is selected, the one with the best normalization pearson statistic will be used. Identity is automatically selected and compared.
`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'
HolidayLookback	
	Number of days in range to compute number of holidays from a given date in the data. If NULL, the number of days are computed for you.
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
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
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
```

Core parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#core-parameter

`input_model`	= NULL, # continue training a model that is stored to fil
`task`	'train' or 'refit'
`device_type`	'cpu' or 'gpu'
`objective`	'binary'
`metric`	'binary_logloss', 'average_precision', 'auc', 'map', 'binary_error', 'auc_mu'
`boosting`	'gbdt', 'rf', 'dart', 'goss'
`LinearTree`	FALSE
`Trees`	50L
`eta`	NULL
`num_leaves`	31
`deterministic`	TRUE

Learning Parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#learning-control-parameter

```
force_col_wise FALSE
force_row_wise FALSE
max_depth      NULL
min_data_in_leaf
                20
min_sum_hessian_in_leaf
                0.001
bagging_freq   0
bagging_fraction
                1.0
feature_fraction
                1.0
feature_fraction_bynode
                1.0
extra_trees    FALSE
early_stopping_round
                10
first_metric_only
                TRUE
max_delta_step 0.0
lambda_l1      0.0
lambda_l2      0.0
linear_lambda  0.0
min_gain_to_split
                0
drop_rate_dart 0.10
max_drop_dart  50
```

```
skip_drop_dart 0.50
uniform_drop_dart
                FALSE
top_rate_goss  FALSE
other_rate_goss
                FALSE
monotone_constraints
                "gbdt_prediction.cpp"
monotone_constraints_method
                'advanced'
monotone_penalty
                0.0
forcedsplits_filename
                NULL # use for AutoStack option; .json fil
refit_decay_rate
                0.90
path_smooth    0.0
                # IO Dataset Parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#io-
                parameters
max_bin        255
min_data_in_bin
                3
data_random_seed
                1
is_enable_sparse
                TRUE
enable_bundle  TRUE
use_missing    TRUE
zero_as_missing
                FALSE
two_round      FALSE
                # Convert Parameters # https://lightgbm.readthedocs.io/en/latest/Parameters.html#convert-
                parameters
convert_model  'gbdt_prediction.cpp'
convert_model_language
                'cpp'
                # Objective Parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#objective-
                parameters
boost_from_average
                TRUE
is_unbalance   FALSE
scale_pos_weight
                1.0
                # Metric Parameters (metric is in Core)
is_provide_training_metric
                TRUE
eval_at        c(1,2,3,4,5)
                # Network Parameter
```

num_machines	1
	# GPU Parameter
gpu_platform_id	
	-1
gpu_device_id	-1
gpu_use_dp	TRUE
num_gpu	1
NThreads	only list up to number of cores, not threads. parallel::detectCores() / 2

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(), AutoLightGBMCARMA(), AutoXGBoostCARMA(), AutoXGBoostHurdleCARMA()

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::AutoLightGBMHurdleCARMA(
```

```
# data args
data = data, # TwoGroup_Data,
TargetColumnName = 'Weekly_Sales',
DateColumnName = 'Date',
HierarchGroups = NULL,
GroupVariables = c('Dept'),
EncodingMethod = "credibility",
TimeWeights = 1,
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', 'Sqrt', 'Logit', 'YeoJohnson'),
Difference = FALSE,
NonNegativePred = FALSE,
RoundPreds = FALSE,

# Date features
CalendarVariables = c('week', 'wom', 'month', 'quarter'),
HolidayVariable = c('USPublicHolidays',
  'EasterGroup',
  'ChristmasGroup','OtherEcclesticalFeasts'),
HolidayLookback = NULL,
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,
      MaxRunsWithoutNewWinner = 50,
      MaxRunMinutes = 60*60,

      # Core parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#core-parameters
      input_model = list('classifier' = NULL, 'regression' = NULL),
      task = list('classifier' = 'train', 'regression' = 'train'),
      device_type = list('classifier' = 'CPU', 'regression' = 'CPU'),
      objective = list('classifier' = 'binary', 'regression' = 'regression'),
      metric = list('classifier' = 'binary_logloss', 'regression' = 'rmse'),
      boosting = list('classifier' = 'gbdt', 'regression' = 'gbdt'),
      LinearTree = list('classifier' = FALSE, 'regression' = FALSE),
      Trees = list('classifier' = 1000L, 'regression' = 1000L),
      eta = list('classifier' = NULL, 'regression' = NULL),
      num_leaves = list('classifier' = 31, 'regression' = 31),
      deterministic = list('classifier' = TRUE, 'regression' = TRUE),

      # Learning Parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#learning-control-parameter
      force_col_wise = list('classifier' = FALSE, 'regression' = FALSE),
      force_row_wise = list('classifier' = FALSE, 'regression' = FALSE),
      max_depth = list('classifier' = NULL, 'regression' = NULL),
      min_data_in_leaf = list('classifier' = 20, 'regression' = 20),
      min_sum_hessian_in_leaf = list('classifier' = 0.001, 'regression' = 0.001),
      bagging_freq = list('classifier' = 0, 'regression' = 0),
      bagging_fraction = list('classifier' = 1.0, 'regression' = 1.0),
      feature_fraction = list('classifier' = 1.0, 'regression' = 1.0),
      feature_fraction_bynode = list('classifier' = 1.0, 'regression' = 1.0),
      extra_trees = list('classifier' = FALSE, 'regression' = FALSE),
      early_stopping_round = list('classifier' = 10, 'regression' = 10),
      first_metric_only = list('classifier' = TRUE, 'regression' = TRUE),
      max_delta_step = list('classifier' = 0.0, 'regression' = 0.0),
      lambda_l1 = list('classifier' = 0.0, 'regression' = 0.0),
      lambda_l2 = list('classifier' = 0.0, 'regression' = 0.0),
      linear_lambda = list('classifier' = 0.0, 'regression' = 0.0),
      min_gain_to_split = list('classifier' = 0, 'regression' = 0),
      drop_rate_dart = list('classifier' = 0.10, 'regression' = 0.10),
      max_drop_dart = list('classifier' = 50, 'regression' = 50),
      skip_drop_dart = list('classifier' = 0.50, 'regression' = 0.50),
      uniform_drop_dart = list('classifier' = FALSE, 'regression' = FALSE),
      top_rate_goss = list('classifier' = FALSE, 'regression' = FALSE),
      other_rate_goss = list('classifier' = FALSE, 'regression' = FALSE),
      monotone_constraints = list('classifier' = NULL, 'regression' = NULL),
    monotone_constraints_method = list('classifier' = 'advanced', 'regression' = 'advanced'),
      monotone_penalty = list('classifier' = 0.0, 'regression' = 0.0),
      forcedsplits_filename = list('classifier' = NULL, 'regression' = NULL),
      refit_decay_rate = list('classifier' = 0.90, 'regression' = 0.90),
      path_smooth = list('classifier' = 0.0, 'regression' = 0.0),

      # IO Dataset Parameters
      max_bin = list('classifier' = 255, 'regression' = 255),
      min_data_in_bin = list('classifier' = 3, 'regression' = 3),
      data_random_seed = list('classifier' = 1, 'regression' = 1),
      is_enable_sparse = list('classifier' = TRUE, 'regression' = TRUE),
```

```
    enable_bundle = list('classifier' = TRUE, 'regression' = TRUE),
    use_missing = list('classifier' = TRUE, 'regression' = TRUE),
    zero_as_missing = list('classifier' = FALSE, 'regression' = FALSE),
    two_round = list('classifier' = FALSE, 'regression' = FALSE),

    # Convert Parameters
    convert_model = list('classifier' = NULL, 'regression' = NULL),
    convert_model_language = list('classifier' = "cpp", 'regression' = "cpp"),

    # Objective Parameters
    boost_from_average = list('classifier' = TRUE, 'regression' = TRUE),
    is_unbalance = list('classifier' = FALSE, 'regression' = FALSE),
    scale_pos_weight = list('classifier' = 1.0, 'regression' = 1.0),

    # Metric Parameters (metric is in Core)
    is_provide_training_metric = list('classifier' = TRUE, 'regression' = TRUE),
    eval_at = list('classifier' = c(1,2,3,4,5), 'regression' = c(1,2,3,4,5)),

    # Network Parameters
    num_machines = list('classifier' = 1, 'regression' = 1),

    # GPU Parameters
    gpu_platform_id = list('classifier' = -1, 'regression' = -1),
    gpu_device_id = list('classifier' = -1, 'regression' = -1),
    gpu_use_dp = list('classifier' = TRUE, 'regression' = TRUE),
    num_gpu = list('classifier' = 1, 'regression' = 1))

# 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::AutoLightGBMHurdleCARMA(
```

```
# data args
data = data,
TargetColumnName = 'Weekly_Sales',
DateColumnName = 'Date',
HierarchGroups = NULL,
GroupVariables = c('Store','Dept'),
EncodingMethod = "credibility",
TimeWeights = 1,
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', 'Sqrt', 'Logit', 'YeoJohnson'),
Difference = FALSE,
NonNegativePred = FALSE,
Threshold = NULL,
RoundPreds = FALSE,

# Date features
CalendarVariables = c('week', 'wom', 'month', 'quarter'),
HolidayVariable = c('USPublicHolidays',
                    'EasterGroup',
                    'ChristmasGroup','OtherEcclesticalFeasts'),
HolidayLookback = NULL,
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,
MaxRunsWithoutNewWinner = 50,
MaxRunMinutes = 60*60,

# Core parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#core-parameters
input_model = list('classifier' = NULL, 'regression' = NULL),
task = list('classifier' = 'train', 'regression' = 'train'),
device_type = list('classifier' = 'CPU', 'regression' = 'CPU'),
objective = list('classifier' = 'binary', 'regression' = 'regression'),
metric = list('classifier' = 'binary_logloss', 'regression' = 'rmse'),
boosting = list('classifier' = 'gbdt', 'regression' = 'gbdt'),
LinearTree = list('classifier' = FALSE, 'regression' = FALSE),
Trees = list('classifier' = 1000L, 'regression' = 1000L),
eta = list('classifier' = NULL, 'regression' = NULL),
num_leaves = list('classifier' = 31, 'regression' = 31),
deterministic = list('classifier' = TRUE, 'regression' = TRUE),

# Learning Parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#learning-control-parameter
force_col_wise = list('classifier' = FALSE, 'regression' = FALSE),
force_row_wise = list('classifier' = FALSE, 'regression' = FALSE),
max_depth = list('classifier' = NULL, 'regression' = NULL),
min_data_in_leaf = list('classifier' = 20, 'regression' = 20),
min_sum_hessian_in_leaf = list('classifier' = 0.001, 'regression' = 0.001),
bagging_freq = list('classifier' = 0, 'regression' = 0),
bagging_fraction = list('classifier' = 1.0, 'regression' = 1.0),
feature_fraction = list('classifier' = 1.0, 'regression' = 1.0),
feature_fraction_bynode = list('classifier' = 1.0, 'regression' = 1.0),
extra_trees = list('classifier' = FALSE, 'regression' = FALSE),
early_stopping_round = list('classifier' = 10, 'regression' = 10),
first_metric_only = list('classifier' = TRUE, 'regression' = TRUE),
max_delta_step = list('classifier' = 0.0, 'regression' = 0.0),
lambda_l1 = list('classifier' = 0.0, 'regression' = 0.0),
lambda_l2 = list('classifier' = 0.0, 'regression' = 0.0),
linear_lambda = list('classifier' = 0.0, 'regression' = 0.0),
min_gain_to_split = list('classifier' = 0, 'regression' = 0),
drop_rate_dart = list('classifier' = 0.10, 'regression' = 0.10),
max_drop_dart = list('classifier' = 50, 'regression' = 50),
skip_drop_dart = list('classifier' = 0.50, 'regression' = 0.50),
uniform_drop_dart = list('classifier' = FALSE, 'regression' = FALSE),
top_rate_goss = list('classifier' = FALSE, 'regression' = FALSE),
other_rate_goss = list('classifier' = FALSE, 'regression' = FALSE),
monotone_constraints = list('classifier' = NULL, 'regression' = NULL),
monotone_constraints_method = list('classifier' = 'advanced', 'regression' = 'advanced'),
monotone_penalty = list('classifier' = 0.0, 'regression' = 0.0),
forcedsplits_filename = list('classifier' = NULL, 'regression' = NULL),
refit_decay_rate = list('classifier' = 0.90, 'regression' = 0.90),
path_smooth = list('classifier' = 0.0, 'regression' = 0.0),

# IO Dataset Parameters
max_bin = list('classifier' = 255, 'regression' = 255),
min_data_in_bin = list('classifier' = 3, 'regression' = 3),
data_random_seed = list('classifier' = 1, 'regression' = 1),
```

```
  is_enable_sparse = list('classifier' = TRUE, 'regression' = TRUE),
  enable_bundle = list('classifier' = TRUE, 'regression' = TRUE),
  use_missing = list('classifier' = TRUE, 'regression' = TRUE),
  zero_as_missing = list('classifier' = FALSE, 'regression' = FALSE),
  two_round = list('classifier' = FALSE, 'regression' = FALSE),

  # Convert Parameters
  convert_model = list('classifier' = NULL, 'regression' = NULL),
  convert_model_language = list('classifier' = "cpp", 'regression' = "cpp"),

  # Objective Parameters
  boost_from_average = list('classifier' = TRUE, 'regression' = TRUE),
  is_unbalance = list('classifier' = FALSE, 'regression' = FALSE),
  scale_pos_weight = list('classifier' = 1.0, 'regression' = 1.0),

  # Metric Parameters (metric is in Core)
  is_provide_training_metric = list('classifier' = TRUE, 'regression' = TRUE),
  eval_at = list('classifier' = c(1,2,3,4,5), 'regression' = c(1,2,3,4,5)),

  # Network Parameters
  num_machines = list('classifier' = 1, 'regression' = 1),

  # GPU Parameters
  gpu_platform_id = list('classifier' = -1, 'regression' = -1),
  gpu_device_id = list('classifier' = -1, 'regression' = -1),
  gpu_use_dp = list('classifier' = TRUE, 'regression' = TRUE),
  num_gpu = list('classifier' = 1, 'regression' = 1))

## End(Not run)
```

AutoLightGBMHurdleModel
                        AutoLightGBMHurdleModel

Description

AutoLightGBMHurdleModel is generalized hurdle modeling framework

Usage

```
AutoLightGBMHurdleModel(
  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,
```

```
PrimaryDateColumn = NULL,
WeightsColumnName = NULL,
ClassWeights = c(1, 1),
IDcols = NULL,
DebugMode = FALSE,
EncodingMethod = "credibility",
TransformNumericColumns = NULL,
Methods = c("Asinh", "Asin", "Log", "LogPlus1", "Sqrt", "Logit"),
SplitRatios = c(0.7, 0.2, 0.1),
SaveModelObjects = FALSE,
ReturnModelObjects = TRUE,
NumOfParDepPlots = 1L,
GridTune = FALSE,
grid_eval_metric = "accuracy",
MaxModelsInGrid = 1L,
BaselineComparison = "default",
MaxRunsWithoutNewWinner = 10L,
MaxRunMinutes = 60L,
input_model = list(classifier = NULL, regression = NULL),
task = list(classifier = "train", regression = "train"),
device_type = list(classifier = "CPU", regression = "CPU"),
objective = list(classifier = "binary", regression = "regression"),
metric = list(classifier = "binary_logloss", regression = "rmse"),
boosting = list(classifier = "gbdt", regression = "gbdt"),
LinearTree = list(classifier = FALSE, regression = FALSE),
Trees = list(classifier = 1000L, regression = 1000L),
eta = list(classifier = NULL, regression = NULL),
num_leaves = list(classifier = 31, regression = 31),
deterministic = list(classifier = TRUE, regression = TRUE),
force_col_wise = list(classifier = FALSE, regression = FALSE),
force_row_wise = list(classifier = FALSE, regression = FALSE),
max_depth = list(classifier = NULL, regression = NULL),
min_data_in_leaf = list(classifier = 20, regression = 20),
min_sum_hessian_in_leaf = list(classifier = 0.001, regression = 0.001),
bagging_freq = list(classifier = 0, regression = 0),
bagging_fraction = list(classifier = 1, regression = 1),
feature_fraction = list(classifier = 1, regression = 1),
feature_fraction_bynode = list(classifier = 1, regression = 1),
extra_trees = list(classifier = FALSE, regression = FALSE),
early_stopping_round = list(classifier = 10, regression = 10),
first_metric_only = list(classifier = TRUE, regression = TRUE),
max_delta_step = list(classifier = 0, regression = 0),
lambda_l1 = list(classifier = 0, regression = 0),
lambda_l2 = list(classifier = 0, regression = 0),
linear_lambda = list(classifier = 0, regression = 0),
min_gain_to_split = list(classifier = 0, regression = 0),
drop_rate_dart = list(classifier = 0.1, regression = 0.1),
max_drop_dart = list(classifier = 50, regression = 50),
skip_drop_dart = list(classifier = 0.5, regression = 0.5),
uniform_drop_dart = list(classifier = FALSE, regression = FALSE),
top_rate_goss = list(classifier = FALSE, regression = FALSE),
other_rate_goss = list(classifier = FALSE, regression = FALSE),
```

```
    monotone_constraints = list(classifier = NULL, regression = NULL),
  monotone_constraints_method = list(classifier = "advanced", regression = "advanced"),
   monotone_penalty = list(classifier = 0, regression = 0),
   forcedsplits_filename = list(classifier = NULL, regression = NULL),
   refit_decay_rate = list(classifier = 0.9, regression = 0.9),
   path_smooth = list(classifier = 0, regression = 0),
   max_bin = list(classifier = 255, regression = 255),
   min_data_in_bin = list(classifier = 3, regression = 3),
   data_random_seed = list(classifier = 1, regression = 1),
   is_enable_sparse = list(classifier = TRUE, regression = TRUE),
   enable_bundle = list(classifier = TRUE, regression = TRUE),
   use_missing = list(classifier = TRUE, regression = TRUE),
   zero_as_missing = list(classifier = FALSE, regression = FALSE),
   two_round = list(classifier = FALSE, regression = FALSE),
   convert_model = list(classifier = NULL, regression = NULL),
   convert_model_language = list(classifier = "cpp", regression = "cpp"),
   boost_from_average = list(classifier = TRUE, regression = TRUE),
   is_unbalance = list(classifier = FALSE, regression = FALSE),
   scale_pos_weight = list(classifier = 1, regression = 1),
   is_provide_training_metric = list(classifier = TRUE, regression = TRUE),
   eval_at = list(classifier = c(1, 2, 3, 4, 5), regression = c(1, 2, 3, 4, 5)),
   num_machines = list(classifier = 1, regression = 1),
   gpu_platform_id = list(classifier = -1, regression = -1),
   gpu_device_id = list(classifier = -1, regression = -1),
   gpu_use_dp = list(classifier = TRUE, regression = TRUE),
   num_gpu = list(classifier = 1, regression = 1)
)
```

Arguments

`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 # Core parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#core-parameter
`NThreads`	only list up to number of cores, not threads. parallel::detectCores() / 2
`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)

PrimaryDateColumn
            Date column for sorting

WeightsColumnName
            Weighs column name

ClassWeights   Look up the classifier model help file

IDcols   Includes PrimaryDateColumn and any other columns you want returned in the
                  validation data with predictions

DebugMode   For debugging

EncodingMethod   Choose from 'binary', 'poly_encode', 'backward_difference', 'helmert' for mul-
                  ticlass cases and additionally 'm_estimator', 'credibility', 'woe', 'target_encoding'
                  for classification use cases.

TransformNumericColumns
            Transform numeric column inside the AutoCatBoostRegression() function

Methods   Choose from 'Asinh', 'Asin', 'Log', 'LogPlus1', 'Sqrt', 'Logit'

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

input_model   = NULL, # continue training a model that is stored to fil

task   'train' or 'refit'

device_type   'cpu' or 'gpu'

objective   'binary'

metric   'binary_logloss', 'average_precision', 'auc', 'map', 'binary_error', 'auc_mu'

boosting   'gbdt', 'rf', 'dart', 'goss'

LinearTree   FALSE

Trees   50L

eta   NULL

num_leaves   31

```
deterministic    TRUE
                 # Learning Parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#learning-
                 control-parameter
force_col_wise   FALSE
force_row_wise   FALSE
max_depth        NULL
min_data_in_leaf
                 20
min_sum_hessian_in_leaf
                 0.001
bagging_freq     0
bagging_fraction
                 1.0
feature_fraction
                 1.0
feature_fraction_bynode
                 1.0
extra_trees      FALSE
early_stopping_round
                 10
first_metric_only
                 TRUE
max_delta_step   0.0
lambda_l1        0.0
lambda_l2        0.0
linear_lambda    0.0
min_gain_to_split
                 0
drop_rate_dart   0.10
max_drop_dart    50
skip_drop_dart   0.50
uniform_drop_dart
                 FALSE
top_rate_goss    FALSE
other_rate_goss
                 FALSE
monotone_constraints
                 "gbdt_prediction.cpp"
monotone_constraints_method
                 'advanced'
monotone_penalty
                 0.0
forcedsplits_filename
                 NULL # use for AutoStack option; .json fil
refit_decay_rate
                 0.90
```

`path_smooth`	0.0
	# IO Dataset Parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#io-parameters
`max_bin`	255
`min_data_in_bin`	
	3
`data_random_seed`	
	1
`is_enable_sparse`	
	TRUE
`enable_bundle`	TRUE
`use_missing`	TRUE
`zero_as_missing`	
	FALSE
`two_round`	FALSE
	# Convert Parameters # https://lightgbm.readthedocs.io/en/latest/Parameters.html#convert-parameters
`convert_model`	'gbdt_prediction.cpp'
`convert_model_language`	
	'cpp'
	# Objective Parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#objective-parameters
`boost_from_average`	
	TRUE
`is_unbalance`	FALSE
`scale_pos_weight`	
	1.0
	# Metric Parameters (metric is in Core)
`is_provide_training_metric`	
	TRUE
`eval_at`	c(1,2,3,4,5)
	# Network Parameter
`num_machines`	1
	# GPU Parameter
`gpu_platform_id`	
	-1
`gpu_device_id`	-1
`gpu_use_dp`	TRUE
`num_gpu`	1

Author(s)

Adrian Antico

See Also

Other Supervised Learning - Hurdle Modeling: `AutoCatBoostHurdleModel()`, `AutoH2oDRFHurdleModel()`, `AutoH2oGBMHurdleModel()`, `AutoXGBoostHurdleModel()`

Examples

```
## Not run:
# Test data.table
LightGBM_QA <- data.table::CJ(
  TOF = c(TRUE,FALSE),
  Classification = c(TRUE,FALSE),
  Success = "Failure",
  ScoreSuccess = "Failure",
  PartitionInFunction = c(TRUE,FALSE), sorted = FALSE
)

# Remove impossible combinations
LightGBM_QA <- LightGBM_QA[!(PartitionInFunction & TOF)]
LightGBM_QA[, RunNumber := seq_len(.N)]

# Path File
Path <- getwd()

#       TOF Classification Success PartitionInFunction RunNumber
# 1:  TRUE           TRUE Failure               FALSE         1
# 2:  TRUE          FALSE Failure               FALSE         2
# 3: FALSE           TRUE Failure                TRUE         3
# 4: FALSE           TRUE Failure               FALSE         4
# 5: FALSE          FALSE Failure                TRUE         5
# 6: FALSE          FALSE Failure               FALSE         6

# AutoCatBoostHurdleModel
# run = 1
# run = 6
for(run in seq_len(LightGBM_QA[,.N])) {

  # Define values
  tof <- LightGBM_QA[run, TOF]
  PartitionInFunction <- LightGBM_QA[run, PartitionInFunction]
  Classify <- LightGBM_QA[run, Classification]
  Tar <- "Adrian"

  # Get data
  if(Classify) {
    data <- RemixAutoML::FakeDataGenerator(N = 15000, ZIP = 1)
  } else {
    data <- RemixAutoML::FakeDataGenerator(N = 100000, ZIP = 2)
  }

  # Partition Data
  if(!tof && !PartitionInFunction) {
    Sets <- RemixAutoML::AutoDataPartition(
      data = data,
      NumDataSets = 3,
      Ratios = c(0.7,0.2,0.1),
      PartitionType = "random",
      StratifyColumnNames = "Adrian",
      TimeColumnName = NULL)
    TTrainData <- Sets$TrainData
    VValidationData <- Sets$ValidationData
    TTestData <- Sets$TestData
```

```
  rm(Sets)
} else {
  TTrainData <- data.table::copy(data)
  VValidationData <- NULL
  TTestData <- NULL
}

# Run function
TestModel <- tryCatch({RemixAutoML::AutoLightGBMHurdleModel(

  # Operationalization
  ModelID = 'ModelTest',
  SaveModelObjects = FALSE,
  ReturnModelObjects = TRUE,
  NThreads = parallel::detectCores(),

  # Data related args
  data = TTrainData,
  ValidationData = VValidationData,
  PrimaryDateColumn = "DateTime",
  TestData = TTestData,
  WeightsColumnName = NULL,
  TrainOnFull = tof,
  Buckets = if(Classify) 0L else c(0,2,3),
  TargetColumnName = "Adrian",
 FeatureColNames = names(TTrainData)[!names(data) %in% c("Adrian","IDcol_1","IDcol_2","IDcol_3","IDcol_4",
  IDcols = c("IDcol_1","IDcol_2","IDcol_3","IDcol_4","IDcol_5","DateTime"),
  DebugMode = TRUE,

  # Metadata args
  EncodingMethod = "credibility",
  Paths = getwd(),
  MetaDataPaths = NULL,
  TransformNumericColumns = NULL,
  Methods = c('Asinh', 'Asin', 'Log', 'LogPlus1', 'Logit'),
  ClassWeights = c(1,1),
  SplitRatios = if(PartitionInFunction) c(0.70, 0.20, 0.10) else NULL,
  NumOfParDepPlots = 10L,

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

  # LightGBM parameters
  task = list('classifier' = 'train', 'regression' = 'train'),
  device_type = list('classifier' = 'CPU', 'regression' = 'CPU'),
 objective = if(Classify) list('classifier' = 'binary', 'regression' = 'regression') else list('classifier' =
  metric = if(Classify

) list('classifier' = 'binary_logloss', 'regression' = 'rmse') else list('classifier' = 'multi_logloss', 're
  boosting = list('classifier' = 'gbdt', 'regression' = 'gbdt'),
  LinearTree = list('classifier' = FALSE, 'regression' = FALSE),
  Trees = list('classifier' = 50L, 'regression' = 50L),
```

```
                eta = list('classifier' = NULL, 'regression' = NULL),
                num_leaves = list('classifier' = 31, 'regression' = 31),
                deterministic = list('classifier' = TRUE, 'regression' = TRUE),

                # Learning Parameters
                force_col_wise = list('classifier' = FALSE, 'regression' = FALSE),
                force_row_wise = list('classifier' = FALSE, 'regression' = FALSE),
                max_depth = list('classifier' = NULL, 'regression' = NULL),
                min_data_in_leaf = list('classifier' = 20, 'regression' = 20),
                min_sum_hessian_in_leaf = list('classifier' = 0.001, 'regression' = 0.001),
                bagging_freq = list('classifier' = 0, 'regression' = 0),
                bagging_fraction = list('classifier' = 1.0, 'regression' = 1.0),
                feature_fraction = list('classifier' = 1.0, 'regression' = 1.0),
                feature_fraction_bynode = list('classifier' = 1.0, 'regression' = 1.0),
                extra_trees = list('classifier' = FALSE, 'regression' = FALSE),
                early_stopping_round = list('classifier' = 10, 'regression' = 10),
                first_metric_only = list('classifier' = TRUE, 'regression' = TRUE),
                max_delta_step = list('classifier' = 0.0, 'regression' = 0.0),
                lambda_l1 = list('classifier' = 0.0, 'regression' = 0.0),
                lambda_l2 = list('classifier' = 0.0, 'regression' = 0.0),
                linear_lambda = list('classifier' = 0.0, 'regression' = 0.0),
                min_gain_to_split = list('classifier' = 0, 'regression' = 0),
                drop_rate_dart = list('classifier' = 0.10, 'regression' = 0.10),
                max_drop_dart = list('classifier' = 50, 'regression' = 50),
                skip_drop_dart = list('classifier' = 0.50, 'regression' = 0.50),
                uniform_drop_dart = list('classifier' = FALSE, 'regression' = FALSE),
                top_rate_goss = list('classifier' = FALSE, 'regression' = FALSE),
                other_rate_goss = list('classifier' = FALSE, 'regression' = FALSE),
                monotone_constraints = list('classifier' = NULL, 'regression' = NULL),
            monotone_constraints_method = list('classifier' = 'advanced', 'regression' = 'advanced'),
                monotone_penalty = list('classifier' = 0.0, 'regression' = 0.0),
                forcedsplits_filename = list('classifier' = NULL, 'regression' = NULL),
                refit_decay_rate = list('classifier' = 0.90, 'regression' = 0.90),
                path_smooth = list('classifier' = 0.0, 'regression' = 0.0),

                # IO Dataset Parameters
                max_bin = list('classifier' = 255, 'regression' = 255),
                min_data_in_bin = list('classifier' = 3, 'regression' = 3),
                data_random_seed = list('classifier' = 1, 'regression' = 1),
                is_enable_sparse = list('classifier' = TRUE, 'regression' = TRUE),
                enable_bundle = list('classifier' = TRUE, 'regression' = TRUE),
                use_missing = list('classifier' = TRUE, 'regression' = TRUE),
                zero_as_missing = list('classifier' = FALSE, 'regression' = FALSE),
                two_round = list('classifier' = FALSE, 'regression' = FALSE),

                # Convert Parameters
                convert_model = list('classifier' = NULL, 'regression' = NULL),
                convert_model_language = list('classifier' = "cpp", 'regression' = "cpp"),

                # Objective Parameters
                boost_from_average = list('classifier' = TRUE, 'regression' = TRUE),
                is_unbalance = list('classifier' = FALSE, 'regression' = FALSE),
                scale_pos_weight = list('classifier' = 1.0, 'regression' = 1.0),

                # Metric Parameters (metric is in Core)
                is_provide_training_metric = list('classifier' = TRUE, 'regression' = TRUE),
                eval_at = list('classifier' = c(1,2,3,4,5), 'regression' = c(1,2,3,4,5)),
```

```
    # Network Parameters
    num_machines = list('classifier' = 1, 'regression' = 1),

    # GPU Parameters
    gpu_platform_id = list('classifier' = -1, 'regression' = -1),
    gpu_device_id = list('classifier' = -1, 'regression' = -1),
    gpu_use_dp = list('classifier' = TRUE, 'regression' = TRUE),
    num_gpu = list('classifier' = 1, 'regression' = 1))}, error = function(x) NULL)

  # Outcome
  if(!is.null(TestModel)) LightGBM_QA[run, Success := "Success"]
 data.table::fwrite(LightGBM_QA, file = "C:/Users/Bizon/Documents/GitHub/QA_Code/QA_CSV/AutoLightGBMHurdleM

  # Remove Target Variable
  TTrainData[, c("Target_Buckets", "Adrian") := NULL]

  # Score CatBoost Hurdle Model
  Output <- tryCatch({RemixAutoML::AutoLightGBMHurdleModelScoring(
    TestData = TTrainData,
    Path = Path,
    ModelID = "ModelTest",
    ModelList = TestModel$ModelList,
    ArgsList = TestModel$ArgsList,
    Threshold = NULL)}, error = function(x) NULL)

  # Outcome
  if(!is.null(Output)) LightGBM_QA[run, Score := "Success"]
  TestModel <- NULL
  Output <- NULL
  TTrainData <- NULL
  VValidationData <- NULL
  TTestData <- NULL
  gc(); Sys.sleep(5)
 data.table::fwrite(LightGBM_QA, file = file.path(Path, "AutoLightGBMHurdleModel_QA.csv"))
}

## End(Not run)
```

AutoLightGBMHurdleModelScoring
                    AutoLightGBMHurdleModelScoring

Description

AutoLightGBMHurdleModelScoring can score AutoLightGBMHurdleModel() models

Usage

```
AutoLightGBMHurdleModelScoring(
  TestData = NULL,
  Path = NULL,
  ModelID = NULL,
  ArgsList = NULL,
```

```
    ModelList = NULL,
    Threshold = NULL,
    CARMA = FALSE
)
```

Arguments

TestData	scoring data.table
Path	Supply if ArgsList is NULL or ModelList is null.
ModelID	Supply if ArgsList is NULL or ModelList is null. Same as used in model training.
ArgsList	Output from the hurdle model
ModelList	Output from the hurdle model
Threshold	NULL to use raw probabilities to predict. Otherwise, supply a threshold
CARMA	Keep FALSE. Used for CARMA functions internals

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 Hurdle Modeling: [AutoCatBoostHurdleModelScoring](), [AutoXGBoostHurdleModelScoring

Examples

```
## Not run:

# Define file path
Path <- getwd()

# 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) %in%
  c("Adrian","IDcol_1","IDcol_2","IDcol_3","DateTime")]

Output <- RemixAutoML::AutoLightGBMHurdleModel(

  # Operationalization args
  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,
      PrimaryDateColumn = NULL,
      WeightsColumnName = NULL,
      IDcols = NULL,
      ClassWeights = c(1,1),
      DebugMode = FALSE,

      # options
      EncodingMethod = "credibility",
      TransformNumericColumns = NULL,
      Methods = c('Asinh','Asin','Log','LogPlus1','Sqrt','Logit'),
      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,

   # Core parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#core-parameters
      input_model = list('classifier' = NULL, 'regression' = NULL),
      task = list('classifier' = 'train', 'regression' = 'train'),
      device_type = list('classifier' = 'CPU', 'regression' = 'CPU'),
      objective = list('classifier' = 'binary', 'regression' = 'regression'),
      metric = list('classifier' = 'binary_logloss', 'regression' = 'rmse'),
      boosting = list('classifier' = 'gbdt', 'regression' = 'gbdt'),
      LinearTree = list('classifier' = FALSE, 'regression' = FALSE),
      Trees = list('classifier' = 1000L, 'regression' = 1000L),
      eta = list('classifier' = NULL, 'regression' = NULL),
      num_leaves = list('classifier' = 31, 'regression' = 31),
      deterministic = list('classifier' = TRUE, 'regression' = TRUE),

   # Learning Parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#learning-control-parameter
      force_col_wise = list('classifier' = FALSE, 'regression' = FALSE),
      force_row_wise = list('classifier' = FALSE, 'regression' = FALSE),
      max_depth = list('classifier' = NULL, 'regression' = NULL),
      min_data_in_leaf = list('classifier' = 20, 'regression' = 20),
      min_sum_hessian_in_leaf = list('classifier' = 0.001, 'regression' = 0.001),
```

```
bagging_freq = list('classifier' = 0, 'regression' = 0),
bagging_fraction = list('classifier' = 1.0, 'regression' = 1.0),
feature_fraction = list('classifier' = 1.0, 'regression' = 1.0),
feature_fraction_bynode = list('classifier' = 1.0, 'regression' = 1.0),
extra_trees = list('classifier' = FALSE, 'regression' = FALSE),
early_stopping_round = list('classifier' = 10, 'regression' = 10),
first_metric_only = list('classifier' = TRUE, 'regression' = TRUE),
max_delta_step = list('classifier' = 0.0, 'regression' = 0.0),
lambda_l1 = list('classifier' = 0.0, 'regression' = 0.0),
lambda_l2 = list('classifier' = 0.0, 'regression' = 0.0),
linear_lambda = list('classifier' = 0.0, 'regression' = 0.0),
min_gain_to_split = list('classifier' = 0, 'regression' = 0),
drop_rate_dart = list('classifier' = 0.10, 'regression' = 0.10),
max_drop_dart = list('classifier' = 50, 'regression' = 50),
skip_drop_dart = list('classifier' = 0.50, 'regression' = 0.50),
uniform_drop_dart = list('classifier' = FALSE, 'regression' = FALSE),
top_rate_goss = list('classifier' = FALSE, 'regression' = FALSE),
other_rate_goss = list('classifier' = FALSE, 'regression' = FALSE),
monotone_constraints = list('classifier' = NULL, 'regression' = NULL),
monotone_constraints_method = list('classifier' = 'advanced', 'regression' = 'advanced'),
monotone_penalty = list('classifier' = 0.0, 'regression' = 0.0),
forcedsplits_filename = list('classifier' = NULL, 'regression' = NULL),
refit_decay_rate = list('classifier' = 0.90, 'regression' = 0.90),
path_smooth = list('classifier' = 0.0, 'regression' = 0.0),

# IO Dataset Parameters
max_bin = list('classifier' = 255, 'regression' = 255),
min_data_in_bin = list('classifier' = 3, 'regression' = 3),
data_random_seed = list('classifier' = 1, 'regression' = 1),
is_enable_sparse = list('classifier' = TRUE, 'regression' = TRUE),
enable_bundle = list('classifier' = TRUE, 'regression' = TRUE),
use_missing = list('classifier' = TRUE, 'regression' = TRUE),
zero_as_missing = list('classifier' = FALSE, 'regression' = FALSE),
two_round = list('classifier' = FALSE, 'regression' = FALSE),

# Convert Parameters
convert_model = list('classifier' = NULL, 'regression' = NULL),
convert_model_language = list('classifier' = "cpp", 'regression' = "cpp"),

# Objective Parameters
boost_from_average = list('classifier' = TRUE, 'regression' = TRUE),
is_unbalance = list('classifier' = FALSE, 'regression' = FALSE),
scale_pos_weight = list('classifier' = 1.0, 'regression' = 1.0),

# Metric Parameters (metric is in Core)
is_provide_training_metric = list('classifier' = TRUE, 'regression' = TRUE),
eval_at = list('classifier' = c(1,2,3,4,5), 'regression' = c(1,2,3,4,5)),

# Network Parameters
num_machines = list('classifier' = 1, 'regression' = 1),

# GPU Parameters
gpu_platform_id = list('classifier' = -1, 'regression' = -1),
gpu_device_id = list('classifier' = -1, 'regression' = -1),
gpu_use_dp = list('classifier' = TRUE, 'regression' = TRUE),
num_gpu = list('classifier' = 1, 'regression' = 1))
```

```
# Score XGBoost Hurdle Model
HurdleScores <- RemixAutoML::AutoLightGBMHurdleModelScoring(
  TestData = data,
  Path = Path,
  ModelID = "ModelTest",
  ModelList = NULL,
  ArgsList = NULL,
  Threshold = NULL)

## End(Not run)
```

AutoLightGBMMultiClass

AutoLightGBMMultiClass

Description

AutoLightGBMMultiClass is an automated lightgbm 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

```
AutoLightGBMMultiClass(
  data = NULL,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  PrimaryDateColumn = NULL,
  IDcols = NULL,
  WeightsColumnName = NULL,
  CostMatrixWeights = c(1, 0, 0, 1),
  EncodingMethod = "credibility",
 OutputSelection = c("Importances", "EvalPlots", "EvalMetrics", "Score_TrainData"),
  model_path = NULL,
  metadata_path = NULL,
  DebugMode = FALSE,
  SaveInfoToPDF = FALSE,
  ModelID = "TestModel",
  ReturnFactorLevels = TRUE,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  NumOfParDepPlots = 3L,
  Verbose = 0L,
  GridTune = FALSE,
  grid_eval_metric = "microauc",
```

```
BaselineComparison = "default",
MaxModelsInGrid = 10L,
MaxRunsWithoutNewWinner = 20L,
MaxRunMinutes = 24L * 60L,
PassInGrid = NULL,
input_model = NULL,
task = "train",
device_type = "CPU",
NThreads = parallel::detectCores()/2,
objective = "multiclass",
multi_error_top_k = 1,
metric = "multi_logloss",
boosting = "gbdt",
LinearTree = FALSE,
Trees = 50L,
eta = NULL,
num_leaves = 31,
deterministic = TRUE,
force_col_wise = FALSE,
force_row_wise = FALSE,
max_depth = NULL,
min_data_in_leaf = 20,
min_sum_hessian_in_leaf = 0.001,
bagging_freq = 0,
bagging_fraction = 1,
feature_fraction = 1,
feature_fraction_bynode = 1,
extra_trees = FALSE,
early_stopping_round = 10,
first_metric_only = TRUE,
max_delta_step = 0,
lambda_l1 = 0,
lambda_l2 = 0,
linear_lambda = 0,
min_gain_to_split = 0,
drop_rate_dart = 0.1,
max_drop_dart = 50,
skip_drop_dart = 0.5,
uniform_drop_dart = FALSE,
top_rate_goss = FALSE,
other_rate_goss = FALSE,
monotone_constraints = NULL,
monotone_constraints_method = "advanced",
monotone_penalty = 0,
forcedsplits_filename = NULL,
refit_decay_rate = 0.9,
path_smooth = 0,
max_bin = 255,
min_data_in_bin = 3,
data_random_seed = 1,
is_enable_sparse = TRUE,
enable_bundle = TRUE,
```

```
  use_missing = TRUE,
  zero_as_missing = FALSE,
  two_round = FALSE,
  convert_model = NULL,
  convert_model_language = "cpp",
  boost_from_average = TRUE,
  is_unbalance = FALSE,
  scale_pos_weight = 1,
  is_provide_training_metric = TRUE,
  eval_at = c(1, 2, 3, 4, 5),
  num_machines = 1,
  gpu_platform_id = -1,
  gpu_device_id = -1,
  gpu_use_dp = TRUE,
  num_gpu = 1
)
```

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.
TargetColumnName	
	Either supply the target column name OR the column number where the target is located (but not mixed types).
FeatureColNames	
	Either supply the feature column names OR the column number where the target is located (but not mixed types)
PrimaryDateColumn	
	Supply a date or datetime column for catboost to utilize time as its basis for handling categorical features, instead of random shuffling
IDcols	A vector of column names or column numbers to keep in your data but not include in the modeling.
WeightsColumnName	
	Supply a column name for your weights column. Leave NULL otherwise
EncodingMethod	Choose from 'binary', 'm_estimator', 'credibility', 'woe', 'target_encoding', 'poly_encode', 'backward_difference', 'helmert'
OutputSelection	
	You can select what type of output you want returned. Choose from c("Importances", "EvalPlots", "EvalMetrics", "PDFs", "Score_TrainData")
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.
DebugMode	Set to TRUE to get a print out of the steps taken throughout the function
SaveInfoToPDF	Set to TRUE to save model insights to pdf
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

NumOfParDepPlots

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

Verbose   Set to 0 if you want to suppress model evaluation updates in training

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

    "mae", "mape", "rmse", "r2". Case sensitive

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.

    # Core parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#core-parameter

MaxModelsInGrid

    Number of models to test from grid options (243 total possible options)

MaxRunsWithoutNewWinner

    Runs without new winner to end procedure

MaxRunMinutes   In minutes

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

input_model   = NULL, # continue training a model that is stored to fil

task   'train' or 'refit'

device_type   'cpu' or 'gpu'

NThreads   only list up to number of cores, not threads. parallel::detectCores() / 2

objective   'multiclass', 'multiclassova'

multi_error_top_k

    Default 1. Counts a prediction as correct if the chosen label is in the top K labels. K = 1 == multi_error

metric   'multi_logloss', 'multi_error', 'kullback_leibler', 'cross_entropy', 'cross_entropy_lambda'

boosting   'gbdt', 'rf', 'dart', 'goss'

LinearTree   FALSE

Trees   50L

eta   NULL

num_leaves   31

deterministic   TRUE

    # Learning Parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#learning-control-parameter

force_col_wise FALSE

force_row_wise FALSE

```
max_depth       NULL
min_data_in_leaf
                20
min_sum_hessian_in_leaf
                0.001
bagging_freq    0
bagging_fraction
                1.0
feature_fraction
                1.0
feature_fraction_bynode
                1.0
extra_trees     FALSE
early_stopping_round
                10
first_metric_only
                TRUE
max_delta_step  0.0
lambda_l1       0.0
lambda_l2       0.0
linear_lambda   0.0
min_gain_to_split
                0
drop_rate_dart  0.10
max_drop_dart   50
skip_drop_dart  0.50
uniform_drop_dart
                FALSE
top_rate_goss   FALSE
other_rate_goss
                FALSE
monotone_constraints
                "gbdt_prediction.cpp"
monotone_constraints_method
                'advanced'
monotone_penalty
                0.0
forcedsplits_filename
                NULL # use for AutoStack option; .json fil
refit_decay_rate
                0.90
path_smooth     0.0
                # IO Dataset Parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#io-
                parameters
max_bin         255
min_data_in_bin
                3
```

```
data_random_seed
                1
is_enable_sparse
                TRUE
```

enable_bundle   TRUE

use_missing   TRUE

```
zero_as_missing
                FALSE
```

two_round   FALSE

                # Convert Parameters # https://lightgbm.readthedocs.io/en/latest/Parameters.html#convert-parameters

convert_model   'gbdt_prediction.cpp'

```
convert_model_language
                'cpp'
```

                # Objective Parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#objective-parameters

```
boost_from_average
                TRUE
```

is_unbalance   FALSE

```
scale_pos_weight
                1.0
```

                # Metric Parameters (metric is in Core)

```
is_provide_training_metric
                TRUE
```

eval_at   c(1,2,3,4,5)

                # Network Parameter

num_machines   1

                # GPU Parameter

```
gpu_platform_id
                -1
```

gpu_device_id   -1

gpu_use_dp   TRUE

num_gpu   1

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, and GridList

Author(s)

Adrian Antico

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::AutoLightGBMMultiClass(

  # Metadata args
  OutputSelection = c("Importances","EvalPlots","EvalMetrics","Score_TrainData"),
  model_path = normalizePath("./"),
  metadata_path = NULL,
  ModelID = "Test_Model_1",
  NumOfParDepPlots = 3L,
  EncodingMethod = "credibility",
  ReturnFactorLevels = TRUE,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  SaveInfoToPDF = FALSE,
  DebugMode = FALSE,

  # Data args
  data = data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = "Adrian",
  FeatureColNames = names(data)[!names(data) %in% c("IDcol_1", "IDcol_2","Adrian")],
  PrimaryDateColumn = NULL,
  WeightsColumnName = NULL,
  IDcols = c("IDcol_1","IDcol_2"),

  # Grid parameters
  GridTune = FALSE,
  grid_eval_metric = 'microauc',
  BaselineComparison = 'default',
  MaxModelsInGrid = 10L,
  MaxRunsWithoutNewWinner = 20L,
  MaxRunMinutes = 24L*60L,
  PassInGrid = NULL,

  # Core parameters
  # https://lightgbm.readthedocs.io/en/latest/Parameters.html#core-parameters
  input_model = NULL, # continue training a model that is stored to file
  task = "train",
  device_type = 'CPU',
  NThreads = parallel::detectCores() / 2,
  objective = 'multiclass',
  multi_error_top_k = 1,
```

```
metric = 'multi_logloss',
boosting = 'gbdt',
LinearTree = FALSE,
Trees = 50L,
eta = NULL,
num_leaves = 31,
deterministic = TRUE,

# Learning Parameters
# https://lightgbm.readthedocs.io/en/latest/Parameters.html#learning-control-parameters
force_col_wise = FALSE,
force_row_wise = FALSE,
max_depth = NULL,
min_data_in_leaf = 20,
min_sum_hessian_in_leaf = 0.001,
bagging_freq = 0,
bagging_fraction = 1.0,
feature_fraction = 1.0,
feature_fraction_bynode = 1.0,
extra_trees = FALSE,
early_stopping_round = 10,
first_metric_only = TRUE,
max_delta_step = 0.0,
lambda_l1 = 0.0,
lambda_l2 = 0.0,
linear_lambda = 0.0,
min_gain_to_split = 0,
drop_rate_dart = 0.10,
max_drop_dart = 50,
skip_drop_dart = 0.50,
uniform_drop_dart = FALSE,
top_rate_goss = FALSE,
other_rate_goss = FALSE,
monotone_constraints = NULL,
monotone_constraints_method = "advanced",
monotone_penalty = 0.0,
forcedsplits_filename = NULL, # use for AutoStack option; .json file
refit_decay_rate = 0.90,
path_smooth = 0.0,

# IO Dataset Parameters
# https://lightgbm.readthedocs.io/en/latest/Parameters.html#io-parameters
max_bin = 255,
min_data_in_bin = 3,
data_random_seed = 1,
is_enable_sparse = TRUE,
enable_bundle = TRUE,
use_missing = TRUE,
zero_as_missing = FALSE,
two_round = FALSE,

# Convert Parameters
convert_model = NULL,
convert_model_language = "cpp",

# Objective Parameters
# https://lightgbm.readthedocs.io/en/latest/Parameters.html#objective-parameters
```

```
      boost_from_average = TRUE,
      is_unbalance = FALSE,
      scale_pos_weight = 1.0,

      # Metric Parameters (metric is in Core)
      # https://lightgbm.readthedocs.io/en/latest/Parameters.html#metric-parameters
      is_provide_training_metric = TRUE,
      eval_at = c(1,2,3,4,5),

      # Network Parameters
      # https://lightgbm.readthedocs.io/en/latest/Parameters.html#network-parameters
      num_machines = 1,

      # GPU Parameters
      # https://lightgbm.readthedocs.io/en/latest/Parameters.html#gpu-parameters
      gpu_platform_id = -1,
      gpu_device_id = -1,
      gpu_use_dp = TRUE,
      num_gpu = 1)

  ## End(Not run)
```

AutoLightGBMRegression

AutoLightGBMRegression

Description

AutoLightGBMRegression is an automated lightgbm 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

```
AutoLightGBMRegression(
  data = NULL,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  PrimaryDateColumn = NULL,
  WeightsColumnName = NULL,
  IDcols = NULL,
 OutputSelection = c("Importances", "EvalPlots", "EvalMetrics", "Score_TrainData"),
 model_path = NULL,
 metadata_path = NULL,
 DebugMode = FALSE,
 SaveInfoToPDF = FALSE,
```

```
ModelID = "TestModel",
ReturnFactorLevels = TRUE,
ReturnModelObjects = TRUE,
SaveModelObjects = FALSE,
EncodingMethod = "credibility",
TransformNumericColumns = NULL,
Methods = c("Asinh", "Log", "LogPlus1", "Sqrt", "Asin", "Logit"),
Verbose = 0L,
NumOfParDepPlots = 3L,
GridTune = FALSE,
grid_eval_metric = "r2",
BaselineComparison = "default",
MaxModelsInGrid = 10L,
MaxRunsWithoutNewWinner = 20L,
MaxRunMinutes = 24L * 60L,
PassInGrid = NULL,
input_model = NULL,
task = "train",
device_type = "CPU",
NThreads = parallel::detectCores()/2,
objective = "regression",
metric = "rmse",
boosting = "gbdt",
LinearTree = FALSE,
Trees = 50L,
eta = NULL,
num_leaves = 31,
deterministic = TRUE,
force_col_wise = FALSE,
force_row_wise = FALSE,
max_depth = NULL,
min_data_in_leaf = 20,
min_sum_hessian_in_leaf = 0.001,
bagging_freq = 0,
bagging_fraction = 1,
feature_fraction = 1,
feature_fraction_bynode = 1,
extra_trees = FALSE,
early_stopping_round = 10,
first_metric_only = TRUE,
max_delta_step = 0,
lambda_l1 = 0,
lambda_l2 = 0,
linear_lambda = 0,
min_gain_to_split = 0,
drop_rate_dart = 0.1,
max_drop_dart = 50,
skip_drop_dart = 0.5,
uniform_drop_dart = FALSE,
top_rate_goss = FALSE,
other_rate_goss = FALSE,
monotone_constraints = NULL,
```

```
    monotone_constraints_method = "advanced",
    monotone_penalty = 0,
    forcedsplits_filename = NULL,
    refit_decay_rate = 0.9,
    path_smooth = 0,
    max_bin = 255,
    min_data_in_bin = 3,
    data_random_seed = 1,
    is_enable_sparse = TRUE,
    enable_bundle = TRUE,
    use_missing = TRUE,
    zero_as_missing = FALSE,
    two_round = FALSE,
    convert_model = NULL,
    convert_model_language = "cpp",
    boost_from_average = TRUE,
    alpha = 0.9,
    fair_c = 1,
    poisson_max_delta_step = 0.7,
    tweedie_variance_power = 1.5,
    lambdarank_truncation_level = 30,
    is_provide_training_metric = TRUE,
    eval_at = c(1, 2, 3, 4, 5),
    num_machines = 1,
    gpu_platform_id = -1,
    gpu_device_id = -1,
    gpu_use_dp = TRUE,
    num_gpu = 1
)
```

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.
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
WeightsColumnName	
	Supply a column name for your weights column. Leave NULL otherwise
IDcols	A vector of column names or column numbers to keep in your data but not include in the modeling.

OutputSelection

> You can select what type of output you want returned. Choose from c('Importances', 'EvalPlots', 'EvalMetrics', 'PDFs', 'Score_TrainData')

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.

DebugMode   Set to TRUE to get a print out of the steps taken throughout the function

SaveInfoToPDF   Set to TRUE to save model insights to pdf

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

EncodingMethod Choose from 'binary', 'm_estimator', 'credibility', 'woe', 'target_encoding', 'poly_encode', 'backward_difference', 'helmert'

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', 'Sqrt', 'Logit', 'YeoJohnson'. 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.

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

> 'mae', 'mape', 'rmse', 'r2'. Case sensitive

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

> Runs without new winner to end procedure

MaxRunMinutes   In minutes

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

input_model   = NULL, # continue training a model that is stored to fil

> # Core parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#core-parameter

task   'train' or 'refit'

device_type   'cpu' or 'gpu'

NThreads	only list up to number of cores, not threads. parallel::detectCores() / 2
objective	'regression'
metric	'rmse', 'l1', 'l2', 'quantile', 'mape', 'huber', 'fair', 'poisson', 'gamma', 'gamma_deviance', 'tweedie', 'ndcg'
boosting	'gbdt', 'rf', 'dart', 'goss'
LinearTree	FALSE
Trees	50L
eta	NULL
num_leaves	31
deterministic	TRUE
	# Learning Parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#learning-control-parameter
force_col_wise	FALSE
force_row_wise	FALSE
max_depth	NULL
min_data_in_leaf	
	20
min_sum_hessian_in_leaf	
	0.001
bagging_freq	0
bagging_fraction	
	1.0
feature_fraction	
	1.0
feature_fraction_bynode	
	1.0
extra_trees	FALSE
early_stopping_round	
	10
first_metric_only	
	TRUE
max_delta_step	0.0
lambda_l1	0.0
lambda_l2	0.0
linear_lambda	0.0
min_gain_to_split	
	0
drop_rate_dart	0.10
max_drop_dart	50
skip_drop_dart	0.50
uniform_drop_dart	
	FALSE
top_rate_goss	FALSE
other_rate_goss	
	FALSE

monotone_constraints
                NULL, 'gbdt_prediction.cpp'

monotone_constraints_method
                'advanced'

monotone_penalty
                0.0

forcedsplits_filename
                NULL # use for AutoStack option; .json fil

refit_decay_rate
                0.90

path_smooth   0.0
                # IO Dataset Parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#io-parameters

max_bin   255

min_data_in_bin
                3

data_random_seed
                1

is_enable_sparse
                TRUE

enable_bundle   TRUE

use_missing   TRUE

zero_as_missing
                FALSE

two_round   FALSE
                # Convert Parameters # https://lightgbm.readthedocs.io/en/latest/Parameters.html#convert-parameters

convert_model   'gbdt_prediction.cpp'

convert_model_language
                'cpp'
                # Objective Parameters https://lightgbm.readthedocs.io/en/latest/Parameters.html#objective-parameters

boost_from_average
                TRUE

alpha   0.90

fair_c   1.0

poisson_max_delta_step
                0.70

tweedie_variance_power
                1.5

lambdarank_truncation_level
                30
                # Metric Parameters (metric is in Core)

is_provide_training_metric
                TRUE

eval_at   c(1,2,3,4,5)
                # Network Parameter

```
num_machines    1
                # GPU Parameter
gpu_platform_id
                -1
gpu_device_id   -1
gpu_use_dp      TRUE
num_gpu         1
```

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, and GridList

Author(s)

Adrian Antico

See Also

Other Automated Supervised Learning - Regression: AutoCatBoostRegression(), AutoH2oDRFRegression(), AutoH2oGAMRegression(), AutoH2oGBMRegression(), AutoH2oGLMRegression(), AutoH2oMLRegression(), AutoNLS(), 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::AutoLightGBMRegression(

  # Metadata args
  OutputSelection = c('Importances','EvalPlots','EvalMetrics','Score_TrainData'),
  model_path = normalizePath('./'),
  metadata_path = NULL,
  ModelID = 'Test_Model_1',
  NumOfParDepPlots = 3L,
  EncodingMethod = 'credibility',
  ReturnFactorLevels = TRUE,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  SaveInfoToPDF = FALSE,
  DebugMode = FALSE,

  # Data args
  data = data,
```

```
TrainOnFull = FALSE,
ValidationData = NULL,
TestData = NULL,
TargetColumnName = 'Adrian',
FeatureColNames = names(data)[!names(data) %in% c('IDcol_1', 'IDcol_2','Adrian')],
PrimaryDateColumn = NULL,
WeightsColumnName = NULL,
IDcols = c('IDcol_1','IDcol_2'),
TransformNumericColumns = NULL,
Methods = c('Asinh','Asin','Log','LogPlus1','Sqrt','Logit'),

# Grid parameters
GridTune = FALSE,
grid_eval_metric = 'r2',
BaselineComparison = 'default',
MaxModelsInGrid = 10L,
MaxRunsWithoutNewWinner = 20L,
MaxRunMinutes = 24L*60L,
PassInGrid = NULL,

# Core parameters
# https://lightgbm.readthedocs.io/en/latest/Parameters.html#core-parameters
input_model = NULL, # continue training a model that is stored to file
task = 'train',
device_type = 'CPU',
NThreads = parallel::detectCores() / 2,
objective = 'regression',
metric = 'rmse',
boosting = 'gbdt',
LinearTree = FALSE,
Trees = 50L,
eta = NULL,
num_leaves = 31,
deterministic = TRUE,

# Learning Parameters
# https://lightgbm.readthedocs.io/en/latest/Parameters.html#learning-control-parameters
force_col_wise = FALSE,
force_row_wise = FALSE,
max_depth = NULL,
min_data_in_leaf = 20,
min_sum_hessian_in_leaf = 0.001,
bagging_freq = 0,
bagging_fraction = 1.0,
feature_fraction = 1.0,
feature_fraction_bynode = 1.0,
extra_trees = FALSE,
early_stopping_round = 10,
first_metric_only = TRUE,
max_delta_step = 0.0,
lambda_l1 = 0.0,
lambda_l2 = 0.0,
linear_lambda = 0.0,
min_gain_to_split = 0,
drop_rate_dart = 0.10,
max_drop_dart = 50,
skip_drop_dart = 0.50,
```

```
      uniform_drop_dart = FALSE,
      top_rate_goss = FALSE,
      other_rate_goss = FALSE,
      monotone_constraints = NULL,
      monotone_constraints_method = 'advanced',
      monotone_penalty = 0.0,
      forcedsplits_filename = NULL, # use for AutoStack option; .json file
      refit_decay_rate = 0.90,
      path_smooth = 0.0,

      # IO Dataset Parameters
      # https://lightgbm.readthedocs.io/en/latest/Parameters.html#io-parameters
      max_bin = 255,
      min_data_in_bin = 3,
      data_random_seed = 1,
      is_enable_sparse = TRUE,
      enable_bundle = TRUE,
      use_missing = TRUE,
      zero_as_missing = FALSE,
      two_round = FALSE,

      # Convert Parameters
      convert_model = NULL,
      convert_model_language = 'cpp',

      # Objective Parameters
      # https://lightgbm.readthedocs.io/en/latest/Parameters.html#objective-parameters
      boost_from_average = TRUE,
      alpha = 0.90,
      fair_c = 1.0,
      poisson_max_delta_step = 0.70,
      tweedie_variance_power = 1.5,
      lambdarank_truncation_level = 30,

      # Metric Parameters (metric is in Core)
      # https://lightgbm.readthedocs.io/en/latest/Parameters.html#metric-parameters
      is_provide_training_metric = TRUE,
      eval_at = c(1,2,3,4,5),

      # Network Parameters
      # https://lightgbm.readthedocs.io/en/latest/Parameters.html#network-parameters
      num_machines = 1,

      # GPU Parameters
      # https://lightgbm.readthedocs.io/en/latest/Parameters.html#gpu-parameters
      gpu_platform_id = -1,
      gpu_device_id = -1,
      gpu_use_dp = TRUE,
      num_gpu = 1)

  ## End(Not run)
```

AutoLightGBMScoring   *AutoLightGBMScoring*

Description

AutoLightGBMScoring is an automated scoring function that compliments the AutoLightGBM 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

```
AutoLightGBMScoring(
  TargetType = NULL,
  ScoringData = NULL,
  ReturnShapValues = FALSE,
  FeatureColumnNames = NULL,
  IDcols = NULL,
  EncodingMethod = "credibility",
  FactorLevelsList = NULL,
  TargetLevels = NULL,
  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 AutoLightGBMRegression(), AutoLightGBMClassifier() or AutoLightGBMMultiClass()
ScoringData	This is your data.table of features for scoring. Can be a single row or batch.
ReturnShapValues	Not functional yet. The shap values are returned in a way that is slow and incompatible with the existing tools. Working on a better solution.
FeatureColumnNames	Supply either column names or column numbers used in the AutoLightGBM__() function
IDcols	Supply ID column numbers for any metadata you want returned with your predicted values
EncodingMethod	Choose from 'binary', 'm_estimator', 'credibility', 'woe', 'target_encoding', 'poly_encode', 'backward_difference', 'helmert'

`FactorLevelsList`	
	Supply the factor variables' list from DummifyDT()
`TargetLevels`	Supply the target levels output from AutoLightGBMMultiClass() or the scoring function will go looking for it in the file path you supply.
`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 AutoLightGBM__() function
`ModelID`	Supply the model ID used in the AutoLightGBM__() 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()`, `AutoXGBoostScoring()`

Examples

```
## Not run:
Preds <- RemixAutoML::AutoLightGBMScoring(
  TargetType = 'regression',
  ScoringData = data,
  ReturnShapValues = FALSE,
  FeatureColumnNames = 2:12,
  IDcols = NULL,
  EncodingMethod = 'credibility',
  FactorLevelsList = NULL,
  TargetLevels = NULL,
  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)
```

AutoMarketBasketModel   *AutoMarketBasketModel*

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

Other Recommenders: `AutoRecommenderDataCreate()`, `AutoRecommenderScore()`, `AutoRecommenderTrain()`

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*

Description

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

Usage

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

Arguments

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

Value

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

Author(s)

Adrian Antico

See Also

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

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)
```

AutoRecommenderDataCreate

AutoRecommenderDataCreate

Description

AutoRecommenderDataCreate to create data that is prepared for modeling

Usage

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

Arguments

data   This is your transactional data.table. Must include an Entity (typically cus-
                tomer), 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: `AutoMarketBasketModel`(), `AutoRecommenderScore`(), `AutoRecommenderTrain`()

Examples

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

## End(Not run)
```

AutoRecommenderScore   *AutoRecommenderScore*

Description

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

Usage

```
AutoRecommenderScore(
  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: AutoMarketBasketModel(), AutoRecommenderDataCreate(), AutoRecommenderTrain()

Examples

```
## Not run:
Results <- RemixAutoML::AutoRecommenderScore(
  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)
```

AutoRecommenderTrain   *AutoRecommenderTrain*

Description

This function returns the winning model that you pass onto AutoRecommenderScoring

Usage

```
AutoRecommenderTrain(
  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: `AutoMarketBasketModel()`, `AutoRecommenderDataCreate()`, `AutoRecommenderScore()`

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)
```

AutoShapeShap   *AutoShapeShap*

Description

AutoShapeShap will convert your scored shap values from CatBoost

Usage

```
AutoShapeShap(
  ScoringData = NULL,
  Threads = max(1L, parallel::detectCores() - 2L),
  DateColumnName = "Date",
  ByVariableName = "GroupVariable"
)
```

Arguments

ScoringData	Scoring data from AutoCatBoostScoring with classification or regression
Threads	Number of threads to use for the parellel routine
DateColumnName	Name of the date column in scoring data
ByVariableName	Name of your base entity column name

Author(s)

Adrian Antico

See Also

Other Model Evaluation and Interpretation: [CumGainsChart](), [EvalPlot](), [ParDepCalPlots](), [ROCPlot](), [RedYellowGreen](), [ResidualPlots](), [SingleRowShapeShap](), [threshOptim]()

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,
  FilePath = NULL,
  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,
  NumberCores = max(1L, min(4L, parallel::detectCores() - 2L))
)
```

Arguments

data	Source data.table
FilePath	NULL to return nothing. Provide a file path to save the model and xregs if available
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 pro-
> cedure.

MaxNumberModels

> Indicate the maximum number of models to test.

MaxRunTimeMinutes

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

NumberCores   Default max(1L, min(4L, parallel::detectCores()-2L))

Author(s)

Adrian Antico

See Also

Other Automated Time Series: `AutoArfima()`, `AutoBanditNNet()`, `AutoBanditSarima()`, `AutoETS()`,
`AutoTS()`

Examples

```
## Not run:
# Create fake data
data <- RemixAutoML::FakeDataGenerator(TimeSeries = TRUE, TimeSeriesTimeAgg = "days")

# Build model
Output <- RemixAutoML::AutoTBATS(
  data,
  FilePath = NULL,
  TargetVariableName = "Weekly_Sales",
  DateColumnName = "Date",
  TimeAggLevel = "weeks",
  EvaluationMetric = "MAE",
  NumHoldOutPeriods = 5L,
  NumFCPeriods = 5L,
  MaxLags = 5L,
  MaxMovingAverages = 5L,
```

```
  MaxSeasonalPeriods = 1L,
  TrainWeighting = 0.50,
  MaxConsecutiveFails = 12L,
  MaxNumberModels = 100L,
  MaxRunTimeMinutes = 10L,
  NumberCores = max(1L, min(4L, parallel::detectCores()-2L)))

# Output
Output$ForecastPlot
Output$Forecast
Output$PerformanceGrid

## End(Not run)
```

AutoTransformationCreate

AutoTransformationCreate

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", "Sqrt", "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(), AutoDiffLagN(), AutoHierarchicalFourier(),
AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(),
AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring(), CategoricalEncoding(),
CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(),
H2OAutoencoder(), ModelDataPrep(), TimeSeriesFill()

Examples

```
## Not run:
# Create Fake Data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 25000,
  ID = 2L,
  ZIP = 0,
  FactorCount = 2L,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Columns to transform
Cols <- names(data)[1L:11L]
print(Cols)

# Run function
data <- RemixAutoML::AutoTransformationCreate(
  data,
  ColumnNames = Cols,
 Methods = c("YeoJohnson", "BoxCox", "Asinh", "Log", "LogPlus1", "Sqrt", "Asin", "Logit", "Identity"),
  Path = getwd(),
  TransID = "Trans",
  SaveOutput = TRUE)

## End(Not run)
```

AutoTransformationScore

AutoTransformationScore() is a the complimentary function to Auto-
TransformationCreate()

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()`, `AutoDiffLagN()`, `AutoHierarchicalFourier()`, `AutoInteraction()`, `AutoLagRollMode()`, `AutoLagRollStatsScoring()`, `AutoLagRollStats()`, `AutoTransformationCreate()`, `AutoWord2VecModeler()`, `AutoWord2VecScoring()`, `CategoricalEncoding()`, `CreateCalendarVariables()`, `CreateHolidayVariables()`, `DummifyDT()`, `H2OAutoencoderScoring()`, `H2OAutoencoder()`, `ModelDataPrep()`, `TimeSeriesFill()`

Examples

```
## Not run:
# Create Fake Data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 25000,
  ID = 2L,
  ZIP = 0,
  FactorCount = 2L,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Columns to transform
Cols <- names(data)[1L:11L]
print(Cols)

data <- data[1]
```

```
# Run function
Output <- RemixAutoML::AutoTransformationCreate(
  data,
  ColumnNames = Cols,
 Methods = c("YeoJohnson", "BoxCox", "Asinh", "Log", "LogPlus1", "Sqrt", "Asin", "Logit", "Identity"),
  Path = getwd(),
  TransID = "Model_1",
  SaveOutput = TRUE)

# Output
data <- Output$Data
TransInfo <- Output$FinalResults

# Back Transform
data <- RemixAutoML::AutoTransformationScore(
  data,
  FinalResults = TransInfo,
  Path = NULL,
  TransID = "Model_1")

## End(Not run)
```

AutoTS   *AutoTS*

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:

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

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

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: `AutoArfima()`, `AutoBanditNNet()`, `AutoBanditSarima()`, `AutoETS()`, `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   *AutoWord2VecModeler*

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,
  MinWords = 1,
  WindowSize = 12,
  Epochs = 25,
  SaveModel = "standard",
  Threads = max(1L, parallel::detectCores() - 2L),
  MaxMemory = "28G",
  ModelID = "Model_1"
)
```

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
MinWords	For H2O word2vec model
WindowSize	For H2O word2vec model
Epochs	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
ModelID	Name for saving to file

Author(s)

Adrian Antico

See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoHierarchicalFourier(),
AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(),
AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecScoring(), CategoricalEncoding(),
CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(),
H2OAutoencoder(), ModelDataPrep(), TimeSeriesFill()

Examples

```
## Not run:

# Create fake data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.70,
  N = 1000L,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  AddComment = TRUE,
  ZIP = 2L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Create Model and Vectors
data <- RemixAutoML::AutoWord2VecModeler(
  data,
  BuildType = "individual",
  stringCol = c("Comment"),
  KeepStringCol = FALSE,
  ModelID = "Model_1",
  model_path = getwd(),
  vects = 10,
  MinWords = 1,
  WindowSize = 1,
  Epochs = 25,
  SaveModel = "standard",
  Threads = max(1,parallel::detectCores()-2),
  MaxMemory = "28G")

# Remove data
rm(data)

# Create fake data for mock scoring
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.70,
  N = 1000L,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  AddComment = TRUE,
  ZIP = 2L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
```

```
  MultiClass = FALSE)

# Give h2o a few seconds
Sys.sleep(5L)

# Create vectors for scoring
data <- RemixAutoML::AutoWord2VecScoring(
  data,
  BuildType = 'individual',
  ModelObject = NULL,
  ModelID = "Model_1",
  model_path = getwd(),
  stringCol = "Comment",
  KeepStringCol = FALSE,
  H2OStartUp = TRUE,
  H2OShutdown = TRUE,
  Threads = max(1L, parallel::detectCores() - 2L),
  MaxMemory = "28G")

## End(Not run)
```

AutoWord2VecScoring   *AutoWord2VecScoring*

Description

AutoWord2VecScoring is for scoring models generated by AutoWord2VecModeler()

Usage

```
AutoWord2VecScoring(
  data,
  BuildType = "individual",
  ModelObject = NULL,
  ModelID = "Model_1",
  model_path = NULL,
  stringCol = NULL,
  KeepStringCol = FALSE,
  H2OStartUp = TRUE,
  H2OShutdown = TRUE,
  Threads = max(1L, parallel::detectCores() - 2L),
  MaxMemory = "28G"
)
```

Arguments

data	data.table
BuildType	"individual" or "combined". Used to locate model in file
ModelObject	NULL if you want it loaded in the function
ModelID	Same as in training
model_path	Location of model

stringCol	Columns to transform
KeepStringCol	FALSE to remove string col after creating vectors
H2OStartUp	= TRUE,
Threads	max(1L, parallel::detectCores() - 2L)
MaxMemory	"28G"

Author(s)

Adrian Antico

See Also

Other Feature Engineering: [AutoDataPartition()](), [AutoDiffLagN()](), [AutoHierarchicalFourier()](),
[AutoInteraction()](), [AutoLagRollMode()](), [AutoLagRollStatsScoring()](), [AutoLagRollStats()](),
[AutoTransformationCreate()](), [AutoTransformationScore()](), [AutoWord2VecModeler()](), [CategoricalEncoding()]()
[CreateCalendarVariables()](), [CreateHolidayVariables()](), [DummifyDT()](), [H2OAutoencoderScoring()](),
[H2OAutoencoder()](), [ModelDataPrep()](), [TimeSeriesFill()]()

Examples

```
## Not run:
# Create fake data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.70,
  N = 1000L,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  AddComment = TRUE,
  ZIP = 2L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Create Model and Vectors
data <- RemixAutoML::AutoWord2VecModeler(
  data,
  BuildType = "individual",
  stringCol = c("Comment"),
  KeepStringCol = FALSE,
  ModelID = "Model_1",
  model_path = getwd(),
  vects = 10,
  MinWords = 1,
  WindowSize = 1,
  Epochs = 25,
  SaveModel = "standard",
  Threads = max(1,parallel::detectCores()-2),
  MaxMemory = "28G")

# Remove data
rm(data)

# Create fake data for mock scoring
```

```
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.70,
  N = 1000L,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  AddComment = TRUE,
  ZIP = 2L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Create vectors for scoring
data <- RemixAutoML::AutoWord2VecScoring(
  data,
  BuildType = "individual",
  ModelObject = NULL,
  ModelID = "Model_1",
  model_path = getwd(),
  stringCol = "Comment",
  KeepStringCol = FALSE,
  H2OStartUp = TRUE,
  H2OShutdown = TRUE,
  Threads = max(1L, parallel::detectCores() - 2L),
  MaxMemory = "28G")

## 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: `EDA_Histograms()`, `Mode()`, `PlotGUI()`, `ScatterCopula()`, `UserBaseEvolution()`

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*

Description

AutoXGBoostCARMA 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

```
AutoXGBoostCARMA(
  data = NULL,
  XREGS = NULL,
  NonNegativePred = FALSE,
  RoundPreds = FALSE,
  TrainOnFull = FALSE,
  TargetColumnName = NULL,
  DateColumnName = NULL,
  HierarchGroups = NULL,
  GroupVariables = NULL,
  FC_Periods = 5,
  SaveDataPath = NULL,
  PDFOutputPath = NULL,
  TimeUnit = "week",
  TimeGroups = c("weeks", "months"),
  TargetTransformation = FALSE,
  Methods = c("Asinh", "Log", "LogPlus1", "Sqrt"),
  EncodingMethod = "binary",
  AnomalyDetection = 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 = 0,
 CalendarVariables = c("second", "minute", "hour", "wday", "mday", "yday", "week",
    "wom", "isoweek", "month", "quarter", "year"),
  HolidayVariable = c("USPublicHolidays", "EasterGroup", "ChristmasGroup",
    "OtherEcclesticalFeasts"),
  HolidayLookback = NULL,
  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),
    PartitionType = "random",
    Timer = TRUE,
    DebugMode = FALSE,
    EvalMetric = "MAE",
    LossFunction = "reg:squarederror",
    GridTune = FALSE,
    GridEvalMetric = "mae",
    ModelCount = 30L,
    MaxRunsWithoutNewWinner = 20L,
    MaxRunMinutes = 24L * 60L,
    NTrees = 1000L,
    LearningRate = 0.3,
    MaxDepth = 9L,
    MinChildWeight = 1,
    SubSample = 1,
    ColSampleByTree = 1
)
```

Arguments

data	Supply your full series data set here
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.
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 Group-Variables 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
SaveDataPath	Path to save modeling data
PDFOutputPath	Supply a path to save model insights to PDF
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 'YeoJohnson', 'BoxCox', 'Asinh', 'Log', 'LogPlus1', 'Sqrt', 'Asin', or 'Logit'. If more than one is selected, the one with the best normalization pearson statistic will be used. Identity is automatically selected and compared.
EncodingMethod	Choose from 'binary', 'm_estimator', 'credibility', 'woe', 'target_encoding', 'poly_encode', 'backward_difference', 'helmert'
AnomalyDetection	NULL for not using the service. Other, provide a list, e.g. AnomalyDetection = list('tstat_high' = 4, tstat_low = -4)
Lags	Select the periods for all lag variables you want to create. E.g. c(1:5,52) or list('day' = c(1:10), 'weeks' = c(1:4))
MA_Periods	Select the periods for all moving average variables you want to create. E.g. c(1:5,52) or list('day' = c(2:10), 'weeks' = c(2:4))
SD_Periods	Select the periods for all moving standard deviation variables you want to create. E.g. c(1:5,52) or list('day' = c(2:10), 'weeks' = c(2:4))
Skew_Periods	Select the periods for all moving skewness variables you want to create. E.g. c(1:5,52) or list('day' = c(2:10), 'weeks' = c(2:4))
Kurt_Periods	Select the periods for all moving kurtosis variables you want to create. E.g. c(1:5,52) or list('day' = c(2:10), 'weeks' = c(2:4))
Quantile_Periods	Select the periods for all moving quantiles variables you want to create. E.g. c(1:5,52) or list('day' = c(2:10), 'weeks' = c(2:4))
Quantiles_Selected	Select from the following c('q5','q10','q15','q20','q25','q30','q35','q40','q45','q50','q55','q60',' 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', 'wom', 'isoweek', 'month', 'quarter', 'year'
HolidayVariable	NULL, or select from 'USPublicHolidays', 'EasterGroup', 'ChristmasGroup', 'OtherEcclesticalFeasts'
HolidayLookback	Number of days in range to compute number of holidays from a given date in the data. If NULL, the number of days are computed for you.
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	NULL to do nothing. Otherwise, set to 'maxmax', 'minmax', 'maxmin', 'minmin'. See `TimeSeriesFill` for explanations of each type
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
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
EvalMetric	Select from 'r2', 'RMSE', 'MSE', 'MAE'
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'
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
MaxRunsWithoutNewWinner	
	Number of consecutive runs without a new winner in order to terminate procedure
MaxRunMinutes	Default 24L*60L
NTrees	Select the number of trees you want to have built to train the model
LearningRate	Learning Rate
MaxDepth	Depth
MinChildWeight	Records in leaf
SubSample	Random forecast setting
ColSampleByTree	
	Self explanatory

Value

See examples

Author(s)

Adrian Antico

See Also

Other Automated Panel Data Forecasting: `AutoCatBoostCARMA()`, `AutoCatBoostHurdleCARMA()`, `AutoCatBoostVectorCARMA()`, `AutoH2OCARMA()`, `AutoLightGBMCARMA()`, `AutoLightGBMHurdleCARMA()`, `AutoXGBoostHurdleCARMA()`

Examples

```
## Not run:

# Load data
data <- data.table::fread('https://www.dropbox.com/s/2str3ek4f4cheqi/walmart_train.csv?dl=1')
```

```
# Ensure series have no missing dates (also remove series with more than 25% missing values)
data <- RemixAutoML::TimeSeriesFill(
  data,
  DateColumnName = 'Date',
  GroupVariables = c('Store','Dept'),
  TimeUnit = 'weeks',
  FillType = 'maxmax',
  MaxMissingPercent = 0.25,
  SimpleImpute = TRUE)

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

# Remove IsHoliday column
data[, IsHoliday := NULL]

# Create xregs (this is the include the categorical variables instead of utilizing only the interaction of them)
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))]

 # 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
  EncodingMethod = 'binary',
  ZeroPadSeries = NULL,
  DataTruncate = FALSE,
  SplitRatios = c(1 - 10 / 138, 10 / 138),
  PartitionType = 'timeseries',
  AnomalyDetection = NULL,

  # Productionize
  FC_Periods = 0,
  TrainOnFull = FALSE,
  NThreads = 8,
  Timer = TRUE,
  DebugMode = FALSE,
  SaveDataPath = NULL,
  PDFOutputPath = NULL,

  # Target Transformations
  TargetTransformation = TRUE,
  Methods = c('BoxCox', 'Asinh', 'Asin', 'Log',
```

```
                    'LogPlus1', 'Sqrt', '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,
    Quantiles_Selected = c('q5','q95'),
    XREGS = xregs,
    FourierTerms = 4,
    CalendarVariables = c('week', 'wom', 'month', 'quarter'),
    HolidayVariable = c('USPublicHolidays','EasterGroup',
      'ChristmasGroup','OtherEcclesticalFeasts'),
    HolidayLookback = NULL,
    HolidayLags = 1,
    HolidayMovingAverages = 1:2,
    TimeTrendVariable = TRUE,

    # ML eval args
    TreeMethod = 'hist',
    EvalMetric = 'RMSE',
    LossFunction = 'reg:squarederror',

    # ML grid tuning
    GridTune = FALSE,
    ModelCount = 5,
    MaxRunsWithoutNewWinner = 20L,
    MaxRunMinutes = 24L*60L,

    # ML args
    NTrees = 300,
    LearningRate = 0.3,
    MaxDepth = 9L,
    MinChildWeight = 1.0,
    SubSample = 1.0,
    ColSampleByTree = 1.0)

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*

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(
  OutputSelection = c("Importances", "EvalPlots", "EvalMetrics", "Score_TrainData"),
  data = NULL,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  WeightsColumnName = NULL,
  IDcols = NULL,
  model_path = NULL,
  metadata_path = NULL,
  SaveInfoToPDF = FALSE,
  ModelID = "FirstModel",
  EncodingMethod = "credibility",
  ReturnFactorLevels = TRUE,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  Verbose = 0L,
  NumOfParDepPlots = 3L,
  NThreads = max(1L, parallel::detectCores() - 2L),
  LossFunction = "reg:logistic",
  CostMatrixWeights = c(1, 0, 0, 1),
  grid_eval_metric = "MCC",
  eval_metric = "auc",
  TreeMethod = "hist",
  GridTune = FALSE,
  BaselineComparison = "default",
  MaxModelsInGrid = 10L,
  MaxRunsWithoutNewWinner = 20L,
  MaxRunMinutes = 24L * 60L,
  PassInGrid = NULL,
  Trees = 1000L,
  eta = 0.3,
  max_depth = 9,
  min_child_weight = 1,
  subsample = 1,
  colsample_bytree = 1,
  DebugMode = FALSE
)
```

Arguments

`OutputSelection`	
	You can select what type of output you want returned. Choose from c("Importances", "EvalPlots", "EvalMetrics", "PDFs", "Score_TrainData")
`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)
`WeightsColumnName`	
	Supply a column name for your weights column. Leave NULL otherwise
`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.
`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
`EncodingMethod`	Choose from 'binary', 'm_estimator', 'credibility', 'woe', 'target_encoding', 'poly_encode', 'backward_difference', 'helmert'
`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
`LossFunction`	Select from 'reg:logistic', "binary:logistic"
`CostMatrixWeights`	
	A vector with 4 elements c(True Positive Cost, False Negative Cost, False Positive Cost, True Negative Cost). Default c(1,0,0,1),

grid_eval_metric

        Case sensitive. I typically choose 'Utility' or 'MCC'. Choose from 'Utility', 'MCC', 'Acc', 'F1_Score', 'F2_Score', 'F0.5_Score', 'TPR', 'TNR', 'FNR', 'FPR', 'FDR', 'FOR', 'NPV', 'PPV', 'ThreatScore'

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.

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)

DebugMode   TRUE to print to console the steps taken

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()`, `AutoLightGBMClassifier()`

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 args
  OutputSelection = c('Importances', 'EvalPlots', 'EvalMetrics', 'Score_TrainData'),
  model_path = normalizePath("./"),
  metadata_path = NULL,
  ModelID = "Test_Model_1",
  EncodingMethod = "binary",
  ReturnFactorLevels = TRUE,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  SaveInfoToPDF = FALSE,

  # Data args
  data = data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = "Adrian",
  FeatureColNames = names(data)[!names(data) %in%
    c("IDcol_1", "IDcol_2","Adrian")],
  WeightsColumnName = NULL,
  IDcols = c("IDcol_1","IDcol_2"),

  # Model evaluation
  LossFunction = 'reg:logistic',
  CostMatrixWeights = c(1,0,0,1),
  eval_metric = "auc",
  grid_eval_metric = "MCC",
  NumOfParDepPlots = 3L,

  # Grid tuning args
  PassInGrid = NULL,
```

```
    GridTune = FALSE,
    BaselineComparison = "default",
    MaxModelsInGrid = 10L,
    MaxRunsWithoutNewWinner = 20L,
    MaxRunMinutes = 24L*60L,
    Verbose = 1L,

    # ML args
    Trees = 500L,
    eta = 0.30,
    max_depth = 9L,
    min_child_weight = 1.0,
    subsample = 1,
    colsample_bytree = 1,
    DebugMode = FALSE)

## End(Not run)
```

AutoXGBoostFunnelCARMA

AutoXGBoostFunnelCARMA

Description

AutoXGBoostFunnelCARMA is a forecasting model for cohort funnel forecasting for grouped data or non-grouped data

Usage

```
AutoXGBoostFunnelCARMA(
  data,
  GroupVariables = NULL,
  BaseFunnelMeasure = NULL,
  ConversionMeasure = NULL,
  ConversionRateMeasure = NULL,
  CohortPeriodsVariable = NULL,
  CalendarDate = NULL,
  CohortDate = NULL,
  EncodingMethod = "credibility",
 OutputSelection = c("Importances", "EvalPlots", "EvalMetrics", "Score_TrainData"),
  WeightsColumnName = NULL,
  TruncateDate = NULL,
  PartitionRatios = c(0.7, 0.2, 0.1),
  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,
  DebugMode = FALSE,
 CalendarVariables = c("wday", "mday", "yday", "week", "isoweek", "month", "quarter",
    "year"),
  HolidayGroups = c("USPublicHolidays", "EasterGroup", "ChristmasGroup",
    "OtherEcclesticalFeasts"),
  HolidayLookback = NULL,
  CohortHolidayLags = c(1L, 2L, 7L),
  CohortHolidayMovingAverages = c(3L, 7L),
  CalendarHolidayLags = c(1L, 2L, 7L),
  CalendarHolidayMovingAverages = c(3L, 7L),
  ImputeRollStats = -0.001,
 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,
  GridEvalMetric = "mae",
  NumOfParDepPlots = 1L,
  NThreads = parallel::detectCores(),
  TreeMethod = "hist",
  EvalMetric = "MAE",
  LossFunction = "reg:squarederror",
  Trees = 1000L,
  LearningRate = 0.3,
  MaxDepth = 9L,
  MinChildWeight = 1,
  SubSample = 1,
  ColSampleByTree = 1
)
```

Arguments

`data`	data object
`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
`OutputSelection`	= c('Importances', 'EvalPlots', 'EvalMetrics', 'Score_TrainData')
`WeightsColumnName`	= NULL
`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.
`PartitionRatios`	Requires three values for train, validation, and test data sets
`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.

DebugMode   Internal use

CalendarVariables
                   "wday", "mday", "yday", "week", "isoweek", "month", "quarter", "year"

HolidayGroups   c("USPublicHolidays","EasterGroup","ChristmasGroup","OtherEcclesticalFeasts")

HolidayLookback
                   Number of days in range to compute number of holidays from a given date in
                   the data. If NULL, the number of days are computed for you.

CohortHolidayLags
                   c(1L, 2L, 7L),

CohortHolidayMovingAverages
                   c(3L, 7L),

CalendarHolidayLags
                   c(1L, 2L, 7L),

CalendarHolidayMovingAverages
                   = c(3L, 7L),

ImputeRollStats
                   Constant value to fill NA after running AutoLagRollStats()

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"

Grid tuning

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

ML Args begin

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)

NThreads   = parallel::detectCores()

TreeMethod   Choose from 'hist', 'gpu_hist'

EvalMetric   Select from 'r2', 'RMSE', 'MSE', 'MAE'

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'

Trees   Select the number of trees you want to have built to train the model

LearningRate   Learning Rate

MaxDepth   Depth

MinChildWeight   Records in leaf

SubSample   Random forecast setting

ColSampleByTree

Self explanatory

Author(s)

Adrian Antico

See Also

Other Automated Funnel Data Forecasting: `AutoCatBoostFunnelCARMAScoring()`, `AutoCatBoostFunnelCARMA()`, `AutoLightGBMFunnelCARMAScoring()`, `AutoLightGBMFunnelCARMA()`, `AutoXGBoostFunnelCARMAScoring()`

Examples

```
## Not run:
# Create fake data
data <- RemixAutoML::FakeDataGenerator(ChainLadderData = TRUE)

# Subset data for training
ModelDataBase <- data[CalendarDateColumn < '2020-01-01' & CohortDateColumn < '2020-01-01']
ModelData <- data.table::copy(ModelDataBase)

# Train Funne Model
TestModel <- RemixAutoML::AutoXGBoostFunnelCARMA(

  # Data Arguments
  data = ModelData,
  GroupVariables = NULL,
 BaseFunnelMeasure = "Leads", # if you have XREGS, supply vector such as c("Leads", "XREGS1", "XREGS2")
  ConversionMeasure = "Appointments",
  ConversionRateMeasure = NULL,
  CohortPeriodsVariable = "CohortDays",
  WeightsColumnName = NULL,
  CalendarDate = "CalendarDateColumn",
  CohortDate = "CohortDateColumn",
  PartitionRatios = c(0.70,0.20,0.10),
  TruncateDate = NULL,
  TimeUnit = "days",
  TransformTargetVariable = TRUE,
  TransformMethods = c("Asinh","Asin","Log","LogPlus1","Sqrt","Logit"),
  AnomalyDetection = list(tstat_high = 3, tstat_low = -2),

  # MetaData Arguments
  Jobs = c("eval","train"),
  SaveModelObjects = FALSE,
  ModelID = "ModelTest",
  ModelPath = getwd(),
  MetaDataPath = NULL,
  DebugMode = TRUE,
  NumOfParDepPlots = 1L,
  EncodingMethod = "credibility",
  NThreads = parallel::detectCores(),

  # Feature Engineering Arguments
  CalendarTimeGroups = c("days","weeks","months"),
  CohortTimeGroups = c("days", "weeks"),
  CalendarVariables = c("wday","mday","yday","week","month","quarter","year"),
 HolidayGroups = c("USPublicHolidays","EasterGroup","ChristmasGroup","OtherEcclesticalFeasts"),
  HolidayLookback = NULL,
  CohortHolidayLags = c(1L,2L,7L),
  CohortHolidayMovingAverages = c(3L,7L),
  CalendarHolidayLags = c(1L,2L,7L),
  CalendarHolidayMovingAverages = c(3L,7L),

  # Time Series Features
  ImputeRollStats = -0.001,
  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,12
  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",

  # ML Grid Tuning
  PassInGrid = NULL,
  GridTune = FALSE,
  BaselineComparison = "default",
  MaxModelsInGrid = 25L,
  MaxRunMinutes = 180L,
  MaxRunsWithoutNewWinner = 10L,

  # ML Setup Parameters
  GridEvalMetric = 'mae',

  # XGBoost arguments
  TreeMethod = 'hist',
  EvalMetric = 'MAE',
  LossFunction = 'reg:squarederror',
  Trees = 50L,
  LearningRate = 0.3,
  MaxDepth = 9L,
  MinChildWeight = 1.0,
  SubSample = 1.0,
  ColSampleByTree = 1.0)

# Separate out the Base Funnel Measures Data
LeadsData <- data[, lapply(.SD, data.table::first), .SDcols = c("Leads"), by = c("CalendarDateColumn")]
ModelData <- ModelDataBase[, Leads := NULL]

# Forecast Funnel Model
Test <- RemixAutoML::AutoXGBoostFunnelCARMAScoring(
  TrainData = ModelData,
  ForwardLookingData = LeadsData,
  TrainEndDate = ModelData[, max(CalendarDateColumn)],
  ForecastEndDate = LeadsData[, max(CalendarDateColumn)],
  TrainOutput = TestModel$ModelOutput,
  ArgsList = TestModel$ArgsList,
  ModelPath = NULL,
  MaxCohortPeriod = 15,
  DebugMode = TRUE)

## End(Not run)
```

AutoXGBoostFunnelCARMAScoring

AutoLightGBMFunnelCARMAScoring

Description

AutoLightGBMFunnelCARMAScoring for generating forecasts

Usage

```
AutoXGBoostFunnelCARMAScoring(
  TrainData,
  ForwardLookingData = NULL,
  TrainEndDate = NULL,
  ForecastEndDate = NULL,
  ArgsList = NULL,
  TrainOutput = NULL,
  ModelPath = NULL,
  MaxCohortPeriod = NULL,
  DebugMode = FALSE
)
```

Arguments

TrainData	Data utilized in training. Do not put the BaseFunnelMeasure in this data set. Put it in the ForwardLookingData object
ForwardLookingData	
	Base funnel measure data. Needs to cover the span of the forecast horizon
TrainEndDate	Max date from the training data
ForecastEndDate	
	Max date to forecast out to
ArgsList	Output list from AutoCatBoostFunnelCARMA
TrainOutput	Pass in the model object to speed up forecasting
ModelPath	Path to model location
MaxCohortPeriod	
	Max cohort periods to utilize when forecasting
DebugMode	For debugging issues

Author(s)

Adrian Antico

See Also

Other Automated Funnel Data Forecasting: `AutoCatBoostFunnelCARMAScoring()`, `AutoCatBoostFunnelCARMA()`, `AutoLightGBMFunnelCARMAScoring()`, `AutoLightGBMFunnelCARMA()`, `AutoXGBoostFunnelCARMA()`

Examples

```
## Not run:
# Create fake data
data <- RemixAutoML::FakeDataGenerator(ChainLadderData = TRUE)

# Subset data for training
ModelDataBase <- data[CalendarDateColumn < '2020-01-01' & CohortDateColumn < '2020-01-01']
ModelData <- data.table::copy(ModelDataBase)
```

```
# Train Funne Model
TestModel <- RemixAutoML::AutoXGBoostFunnelCARMA(

  # Data Arguments
  data = ModelData,
  GroupVariables = NULL,
 BaseFunnelMeasure = "Leads", # if you have XREGS, supply vector such as c("Leads", "XREGS1", "XREGS2")
  ConversionMeasure = "Appointments",
  ConversionRateMeasure = NULL,
  CohortPeriodsVariable = "CohortDays",
  WeightsColumnName = NULL,
  CalendarDate = "CalendarDateColumn",
  CohortDate = "CohortDateColumn",
  PartitionRatios = c(0.70,0.20,0.10),
  TruncateDate = NULL,
  TimeUnit = "days",
  TransformTargetVariable = TRUE,
  TransformMethods = c("Asinh","Asin","Log","LogPlus1","Sqrt","Logit"),
  AnomalyDetection = list(tstat_high = 3, tstat_low = -2),

  # MetaData Arguments
  Jobs = c("eval","train"),
  SaveModelObjects = FALSE,
  ModelID = "ModelTest",
  ModelPath = getwd(),
  MetaDataPath = NULL,
  DebugMode = TRUE,
  NumOfParDepPlots = 1L,
  EncodingMethod = "credibility",
  NThreads = parallel::detectCores(),

  # Feature Engineering Arguments
  CalendarTimeGroups = c("days","weeks","months"),
  CohortTimeGroups = c("days", "weeks"),
  CalendarVariables = c("wday","mday","yday","week","month","quarter","year"),
 HolidayGroups = c("USPublicHolidays","EasterGroup","ChristmasGroup","OtherEcclesticalFeasts"),
  HolidayLookback = NULL,
  CohortHolidayLags = c(1L,2L,7L),
  CohortHolidayMovingAverages = c(3L,7L),
  CalendarHolidayLags = c(1L,2L,7L),
  CalendarHolidayMovingAverages = c(3L,7L),

  # Time Series Features
  ImputeRollStats = -0.001,
  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,12
  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",

  # ML Grid Tuning
  PassInGrid = NULL,
  GridTune = FALSE,
  BaselineComparison = "default",
  MaxModelsInGrid = 25L,
  MaxRunMinutes = 180L,
  MaxRunsWithoutNewWinner = 10L,

  # ML Setup Parameters
  GridEvalMetric = 'mae',

  # XGBoost arguments
  TreeMethod = 'hist',
  EvalMetric = 'MAE',
  LossFunction = 'reg:squarederror',
  Trees = 50L,
  LearningRate = 0.3,
  MaxDepth = 9L,
  MinChildWeight = 1.0,
  SubSample = 1.0,
  ColSampleByTree = 1.0)

# Separate out the Base Funnel Measures Data
LeadsData <- data[, lapply(.SD, data.table::first), .SDcols = c("Leads"), by = c("CalendarDateColumn")]
ModelData <- ModelDataBase[, Leads := NULL]

# Forecast Funnel Model
Test <- RemixAutoML::AutoXGBoostFunnelCARMAScoring(
  TrainData = ModelData,
  ForwardLookingData = LeadsData,
  TrainEndDate = ModelData[, max(CalendarDateColumn)],
  ForecastEndDate = LeadsData[, max(CalendarDateColumn)],
  TrainOutput = TestModel$ModelOutput,
  ArgsList = TestModel$ArgsList,
  ModelPath = NULL,
  MaxCohortPeriod = 15,
  DebugMode = TRUE)

## End(Not run)
```

AutoXGBoostHurdleCARMA

AutoXGBoostHurdleCARMA

Description

AutoXGBoostHurdleCARMA 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

```
AutoXGBoostHurdleCARMA(
  data,
  NonNegativePred = FALSE,
  Threshold = NULL,
  RoundPreds = FALSE,
  TrainOnFull = FALSE,
  TargetColumnName = "Target",
  DateColumnName = "DateTime",
  HierarchGroups = NULL,
  GroupVariables = NULL,
  EncodingMethod = "credibility",
  TimeWeights = 1,
  FC_Periods = 30,
  TimeUnit = "week",
  TimeGroups = c("weeks", "months"),
  NumOfParDepPlots = 10L,
  TargetTransformation = FALSE,
  Methods = c("BoxCox", "Asinh", "Log", "LogPlus1", "Sqrt", "Asin", "Logit"),
  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",
    "wom", "isoweek", "month", "quarter", "year"),
  HolidayVariable = c("USPublicHolidays", "EasterGroup", "ChristmasGroup",
    "OtherEcclesticalFeasts"),
  HolidayLookback = NULL,
  HolidayLags = 1L,
  HolidayMovingAverages = 1L:2L,
  TimeTrendVariable = FALSE,
  ZeroPadSeries = NULL,
  DataTruncate = FALSE,
  SplitRatios = c(0.7, 0.2, 0.1),
  PartitionType = "timeseries",
  Timer = TRUE,
  DebugMode = FALSE,
  EvalMetric = "RMSE",
  GridTune = FALSE,
  PassInGrid = NULL,
  ModelCount = 100,
  MaxRunsWithoutNewWinner = 50,
  MaxRunMinutes = 24L * 60L,
  TreeMethod = "hist",
  Trees = list(classifier = 1000, regression = 1000),
```

```
  eta = list(classifier = 0.05, regression = 0.05),
  max_depth = list(classifier = 4L, regression = 4L),
  min_child_weight = list(classifier = 1, regression = 1),
  subsample = list(classifier = 0.55, regression = 0.55),
  colsample_bytree = list(classifier = 0.55, regression = 0.55)
)
```

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 Group-Variables when you have a series for every level of a group or multiple groups.
`EncodingMethod`	Choose from 'binary', 'poly_encode', 'backward_difference', 'helmert' for multiclass cases and additionally 'm_estimator', 'credibility', 'woe', 'target_encoding' for classification use cases.
`TimeWeights`	Timeweights creation. Supply a value, such as 0.9999
`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`	Choose from 'YeoJohnson', 'BoxCox', 'Asinh', 'Log', 'LogPlus1', 'Sqrt', 'Asin', or 'Logit'. If more than one is selected, the one with the best normalization pearson statistic will be used. Identity is automatically selected and compared.
`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'
HolidayLookback	
	Number of days in range to compute number of holidays from a given date in the data. If NULL, the number of days are computed for you.
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
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
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

TreeMethod   "hist" or "gpu_hist"

Trees   = list("classifier" = 1000, "regression" = 1000)

eta   = list("classifier" = 0.05, "regression" = 0.05)

max_depth   = list("classifier" = 4L, "regression" = 4L)

min_child_weight

                  = list("classifier" = 1.0, "regression" = 1.0)

subsample   = list("classifier" = 0.55, "regression" = 0.55)

colsample_bytree

                  = list("classifier" = 0.55, "regression" = 0.55)

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 Tar-
getTransformation 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()`, `AutoLightGBMCARMA()`, `AutoLightGBMHurdleCARMA()`,
`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::AutoXGBoostHurdleCARMA(

 # data args
 data = data, # TwoGroup_Data,
 TargetColumnName = 'Weekly_Sales',
 DateColumnName = 'Date',
 HierarchGroups = NULL,
 GroupVariables = c('Dept'),
 EncodingMethod = "credibility",
 TimeWeights = 1,
 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', 'Sqrt', 'Logit', 'YeoJohnson'),
 Difference = FALSE,
 NonNegativePred = FALSE,
 RoundPreds = FALSE,

 # Date features
 CalendarVariables = c('week', 'wom', 'month', 'quarter'),
 HolidayVariable = c('USPublicHolidays',
   'EasterGroup',
   'ChristmasGroup','OtherEcclesticalFeasts'),
 HolidayLookback = NULL,
 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,
  MaxRunsWithoutNewWinner = 50,
  MaxRunMinutes = 60*60,

  # XGBoost Args
  TreeMethod = "hist",
  Trees = list("classifier" = 1000, "regression" = 1000),
  eta = list("classifier" = 0.05, "regression" = 0.05),
  max_depth = list("classifier" = 4L, "regression" = 4L),
  min_child_weight = list("classifier" = 1.0, "regression" = 1.0),
  subsample = list("classifier" = 0.55, "regression" = 0.55),
  colsample_bytree = list("classifier" = 0.55, "regression" = 0.55))

# 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::AutoXGBoostHurdleCARMA(

  # data args
  data = data,
  TargetColumnName = 'Weekly_Sales',
```

```
DateColumnName = 'Date',
HierarchGroups = NULL,
GroupVariables = c('Store','Dept'),
EncodingMethod = "credibility",
TimeWeights = 1,
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', 'Sqrt', 'Logit', 'YeoJohnson'),
Difference = FALSE,
NonNegativePred = FALSE,
Threshold = NULL,
RoundPreds = FALSE,

# Date features
CalendarVariables = c('week', 'wom', 'month', 'quarter'),
HolidayVariable = c('USPublicHolidays',
                    'EasterGroup',
                    'ChristmasGroup','OtherEcclesticalFeasts'),
HolidayLookback = NULL,
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,
      MaxRunsWithoutNewWinner = 50,
      MaxRunMinutes = 60*60,

      # XGBoost Args
      TreeMethod = "hist",
      Trees = list("classifier" = 1000, "regression" = 1000),
      eta = list("classifier" = 0.05, "regression" = 0.05),
      max_depth = list("classifier" = 4L, "regression" = 4L),
      min_child_weight = list("classifier" = 1.0, "regression" = 1.0),
      subsample = list("classifier" = 0.55, "regression" = 0.55),
      colsample_bytree = list("classifier" = 0.55, "regression" = 0.55))

   ## End(Not run)
```

AutoXGBoostHurdleModel

AutoXGBoostHurdleModel

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,
  PrimaryDateColumn = NULL,
  WeightsColumnName = NULL,
  ClassWeights = c(1, 1),
  IDcols = NULL,
  DebugMode = FALSE,
  EncodingMethod = "credibility",
  TransformNumericColumns = NULL,
  Methods = c("Asinh", "Asin", "Log", "LogPlus1", "Sqrt", "Logit"),
  SplitRatios = c(0.7, 0.2, 0.1),
  SaveModelObjects = FALSE,
  ReturnModelObjects = TRUE,
  NumOfParDepPlots = 1L,
```

```
     GridTune = FALSE,
     grid_eval_metric = "accuracy",
     MaxModelsInGrid = 1L,
     BaselineComparison = "default",
     MaxRunsWithoutNewWinner = 10L,
     MaxRunMinutes = 60L,
     Trees = list(classifier = 1000, regression = 1000),
     eta = list(classifier = 0.05, regression = 0.05),
     max_depth = list(classifier = 4L, regression = 4L),
     min_child_weight = list(classifier = 1, regression = 1),
     subsample = list(classifier = 0.55, regression = 0.55),
     colsample_bytree = list(classifier = 0.55, regression = 0.55)
)
```

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)
PrimaryDateColumn	
	Date column for sorting
WeightsColumnName	
	Weighs column name
ClassWeights	Look up the classifier model help file
IDcols	Includes PrimaryDateColumn and any other columns you want returned in the validation data with predictions
DebugMode	For debugging

EncodingMethod   Choose from 'binary', 'poly_encode', 'backward_difference', 'helmert' for mul-
                  ticlass cases and additionally 'm_estimator', 'credibility', 'woe', 'target_encoding'
                  for classification use cases.

TransformNumericColumns
                  Transform numeric column inside the AutoCatBoostRegression() function

Methods   Choose from 'Asinh', 'Asin', 'Log', 'LogPlus1', 'Sqrt', 'Logit'

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.

Author(s)

Adrian Antico

See Also

Other Supervised Learning - Hurdle Modeling: `AutoCatBoostHurdleModel()`, `AutoH2oDRFHurdleModel()`,
`AutoH2oGBMHurdleModel()`, `AutoLightGBMHurdleModel()`

Examples

```
## Not run:
# Test data.table
XGBoost_QA <- data.table::CJ(
  TOF = c(TRUE,FALSE),
  Classification = c(TRUE,FALSE),
  Success = "Failure",
  ScoreSuccess = "Failure",
  PartitionInFunction = c(TRUE,FALSE), sorted = FALSE
)

# Remove impossible combinations
XGBoost_QA <- XGBoost_QA[!(PartitionInFunction & TOF)]
XGBoost_QA[, RunNumber := seq_len(.N)]

# Path File
Path <- getwd()

#       TOF Classification Success PartitionInFunction RunNumber
# 1:  TRUE           TRUE Failure               FALSE         1
# 2:  TRUE          FALSE Failure               FALSE         2
# 3: FALSE           TRUE Failure                TRUE         3
# 4: FALSE           TRUE Failure               FALSE         4
# 5: FALSE          FALSE Failure                TRUE         5
# 6: FALSE          FALSE Failure               FALSE         6

# AutoCatBoostHurdleModel
# run = 5
# run = 6
for(run in seq_len(XGBoost_QA[,.N])) {

  # Define values
  tof <- XGBoost_QA[run, TOF]
  PartitionInFunction <- XGBoost_QA[run, PartitionInFunction]
  Classify <- XGBoost_QA[run, Classification]
  Tar <- "Adrian"

  # Get data
  if(Classify) {
    data <- RemixAutoML::FakeDataGenerator(N = 15000, ZIP = 1)
  } else {
    data <- RemixAutoML::FakeDataGenerator(N = 100000, ZIP = 2)
  }

  # Partition Data
  if(!tof && !PartitionInFunction) {
    Sets <- RemixAutoML::AutoDataPartition(
      data = data,
      NumDataSets = 3,
      Ratios = c(0.7,0.2,0.1),
      PartitionType = "random",
      StratifyColumnNames = "Adrian",
      TimeColumnName = NULL)
    TTrainData <- Sets$TrainData
    VValidationData <- Sets$ValidationData
    TTestData <- Sets$TestData
```

```
  rm(Sets)
} else {
  TTrainData <- data.table::copy(data)
  VValidationData <- NULL
  TTestData <- NULL
}

# Run function
TestModel <- tryCatch({RemixAutoML::AutoXGBoostHurdleModel(

  # Operationalization
  ModelID = 'ModelTest',
  SaveModelObjects = FALSE,
  ReturnModelObjects = TRUE,
  NThreads = parallel::detectCores(),

  # Data related args
  data = TTrainData,
  ValidationData = VValidationData,
  PrimaryDateColumn = "DateTime",
  TestData = TTestData,
  WeightsColumnName = NULL,
  TrainOnFull = tof,
  Buckets = if(Classify) 0L else c(0,2,3),
  TargetColumnName = "Adrian",
 FeatureColNames = names(TTrainData)[!names(data) %in% c("Adrian","IDcol_1","IDcol_2","IDcol_3","IDcol_4",
  IDcols = c("IDcol_1","IDcol_2","IDcol_3","IDcol_4","IDcol_5","DateTime"),
  DebugMode = TRUE,

  # Metadata args
  EncodingMethod = "credibility",
  Paths = normalizePath('./'),
  MetaDataPaths = NULL,
  TransformNumericColumns = NULL,
  Methods = c('Asinh', 'Asin', 'Log', 'LogPlus1', 'Logit'),
  ClassWeights = c(1,1),
  SplitRatios = if(PartitionInFunction) c(0.70, 0.20, 0.10) else NULL,
  NumOfParDepPlots = 10L,

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

  # XGBoost parameters
  TreeMethod = "hist",
  Trees = list("classifier" = 50, "regression" = 50),
  eta = list("classifier" = 0.05, "regression" = 0.05),
  max_depth = list("classifier" = 4L, "regression" = 4L),
  min_child_weight = list("classifier" = 1.0, "regression" = 1.0),
  subsample = list("classifier" = 0.55, "regression" = 0.55),
 colsample_bytree = list("classifier" = 0.55, "regression" = 0.55))}, error = function(x) NULL)

  # Outcome
```

```
    if(!is.null(TestModel)) XGBoost_QA[run, Success := "Success"]
  data.table::fwrite(XGBoost_QA, file = "C:/Users/Bizon/Documents/GitHub/QA_Code/QA_CSV/AutoXGBoostHurdleMod

    # Remove Target Variable
    TTrainData[, c("Target_Buckets", "Adrian") := NULL]

    # Score XGBoost Hurdle Model
    Output <- tryCatch({RemixAutoML::AutoXGBoostHurdleModelScoring(
      TestData = TTrainData,
      Path = Path,
      ModelID = "ModelTest",
      ModelList = TestModel$ModelList,
      ArgsList = TestModel$ArgsList,
      Threshold = NULL)}, error = function(x) NULL)

    # Outcome
    if(!is.null(Output)) XGBoost_QA[run, Score := "Success"]
    TestModel <- NULL
    Output <- NULL
    TTrainData <- NULL
    VValidationData <- NULL
    TTestData <- NULL
    gc(); Sys.sleep(5)
  data.table::fwrite(XGBoost_QA, file = file.path(Path, "AutoXGBoostHurdleModel_QA.csv"))
}

## End(Not run)
```

AutoXGBoostHurdleModelScoring

AutoXGBoostHurdleModelScoring

Description

AutoXGBoostHurdleModelScoring can score AutoXGBoostHurdleModel() models

Usage

```
AutoXGBoostHurdleModelScoring(
  TestData = NULL,
  Path = NULL,
  ModelID = NULL,
  ArgsList = NULL,
  ModelList = NULL,
  Threshold = NULL,
  CARMA = FALSE
)
```

Arguments

TestData	scoring data.table
Path	Supply if ArgsList is NULL or ModelList is null.

ModelID	Supply if ArgsList is NULL or ModelList is null. Same as used in model training.
ArgsList	Output from the hurdle model
ModelList	Output from the hurdle model
Threshold	NULL to use raw probabilities to predict. Otherwise, supply a threshold
CARMA	Keep FALSE. Used for CARMA functions internals

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 Hurdle Modeling: `AutoCatBoostHurdleModelScoring()`, `AutoLightGBMHurdleModelScoring`

Examples

```
## Not run:

# Define file path
Path <- getwd()

# 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) %in%
  c("Adrian","IDcol_1","IDcol_2","IDcol_3","DateTime")]

Output <- RemixAutoML::AutoXGBoostHurdleModel(

  # Operationalization args
  TrainOnFull = FALSE,
  PassInGrid = NULL,

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

  # data args
  data,
```

```
        ValidationData = NULL,
        TestData = NULL,
        Buckets = 0L,
        TargetColumnName = NULL,
        FeatureColNames = NULL,
        PrimaryDateColumn = NULL,
        WeightsColumnName = NULL,
        IDcols = NULL,
        ClassWeights = c(1,1),
        DebugMode = FALSE,

        # options
        EncodingMethod = "credibility",
        TransformNumericColumns = NULL,
        Methods = c('Asinh','Asin','Log','LogPlus1','Sqrt','Logit'),
        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,

        # XGBoost parameters
        TreeMethod = "hist",
        Trees = list("classifier" = 1000, "regression" = 1000),
        eta = list("classifier" = 0.05, "regression" = 0.05),
        max_depth = list("classifier" = 4L, "regression" = 4L),
        min_child_weight = list("classifier" = 1.0, "regression" = 1.0),
        subsample = list("classifier" = 0.55, "regression" = 0.55),
        colsample_bytree = list("classifier" = 0.55, "regression" = 0.55))

  # Score XGBoost Hurdle Model
  HurdleScores <- RemixAutoML::AutoXGBoostHurdleModelScoring(
    TestData = data,
    Path = Path,
    ModelID = "ModelTest",
    ModelList = NULL,
    ArgsList = NULL,
    Threshold = NULL)

  ## End(Not run)
```

AutoXGBoostMultiClass *AutoXGBoostMultiClass*

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(
 OutputSelection = c("Importances", "EvalPlots", "EvalMetrics", "Score_TrainData"),
 data = NULL,
 TrainOnFull = FALSE,
 ValidationData = NULL,
 TestData = NULL,
 TargetColumnName = NULL,
 FeatureColNames = NULL,
 WeightsColumnName = NULL,
 IDcols = NULL,
 model_path = NULL,
 metadata_path = NULL,
 ModelID = "FirstModel",
 LossFunction = "multi:softprob",
 EncodingMethod = "credibility",
 ReturnFactorLevels = TRUE,
 ReturnModelObjects = TRUE,
 SaveModelObjects = FALSE,
 Verbose = 0L,
 DebugMode = FALSE,
 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,
 Trees = 50L,
 eta = NULL,
 max_depth = NULL,
 min_child_weight = NULL,
 subsample = NULL,
 colsample_bytree = NULL
)
```

Arguments

OutputSelection

You can select what type of output you want returned. Choose from c("Importances", "EvalPlots", "EvalMetrics", "PDFs", "Score_TrainData")

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)
WeightsColumnName	
	Supply a column name for your weights column. Leave NULL otherwise
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
LossFunction	Use 'multi:sofprob', I set it up to return the class label and the individual probabilities, just like catboost. Doesn't come like that off the shelf
EncodingMethod	Choose from 'binary', 'm_estimator', 'credibility', 'woe', 'target_encoding', 'poly_encode', 'backward_difference', 'helmert'
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
DebugMode	Set to TRUE to get a print out of the steps taken internally
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.

Trees   Bandit grid partitioned. Supply a single value for non-grid tuning cases. Other-
                  wise, 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. Oth-
                  erwise, supply a vector for the LearningRate values to test. For running grid tun-
                  ing, 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:
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 args
 OutputSelection = c("Importances", "EvalPlots", "EvalMetrics", "PDFs", "Score_TrainData"),
  model_path = normalizePath("./"),
  metadata_path = normalizePath("./"),
  ModelID = "Test_Model_1",
  EncodingMethod = "binary",
  ReturnFactorLevels = TRUE,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,

  # Data args
  data = data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = "Adrian",
  FeatureColNames = names(data)[!names(data) %in%
                                  c("IDcol_1", "IDcol_2","Adrian")],
  WeightsColumnName = NULL,
  IDcols = c("IDcol_1","IDcol_2"),

  # Model evaluation args
  eval_metric = "merror",
  LossFunction = 'multi:softprob',
  grid_eval_metric = "accuracy",
  NumOfParDepPlots = 3L,

  # Grid tuning args
  PassInGrid = NULL,
  GridTune = FALSE,
  BaselineComparison = "default",
  MaxModelsInGrid = 10L,
  MaxRunsWithoutNewWinner = 20L,
  MaxRunMinutes = 24L*60L,
  Verbose = 1L,
  DebugMode = FALSE,

  # ML args
  Trees = 50L,
  eta = 0.05,
  max_depth = 4L,
  min_child_weight = 1.0,
  subsample = 0.55,
  colsample_bytree = 0.55)

## End(Not run)
```

AutoXGBoostRegression   *AutoXGBoostRegression*

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(
 OutputSelection = c("Importances", "EvalPlots", "EvalMetrics", "Score_TrainData"),
 data = NULL,
 TrainOnFull = FALSE,
 ValidationData = NULL,
 TestData = NULL,
 TargetColumnName = NULL,
 FeatureColNames = NULL,
 PrimaryDateColumn = NULL,
 WeightsColumnName = NULL,
 IDcols = NULL,
 model_path = NULL,
 metadata_path = NULL,
 DebugMode = FALSE,
 SaveInfoToPDF = FALSE,
 ModelID = "FirstModel",
 EncodingMethod = "credibility",
 ReturnFactorLevels = TRUE,
 ReturnModelObjects = TRUE,
 SaveModelObjects = FALSE,
 TransformNumericColumns = NULL,
 Methods = c("Asinh", "Log", "LogPlus1", "Sqrt", "Asin", "Logit"),
 Verbose = 0L,
 NumOfParDepPlots = 3L,
 NThreads = parallel::detectCores(),
 LossFunction = "reg:squarederror",
 eval_metric = "rmse",
 grid_eval_metric = "r2",
 TreeMethod = "hist",
 GridTune = FALSE,
 BaselineComparison = "default",
 MaxModelsInGrid = 10L,
 MaxRunsWithoutNewWinner = 20L,
 MaxRunMinutes = 24L * 60L,
 PassInGrid = NULL,
 Trees = 50L,
 eta = NULL,
 max_depth = NULL,
 min_child_weight = NULL,
 subsample = NULL,
 colsample_bytree = NULL
```

)

Arguments

OutputSelection

You can select what type of output you want returned. Choose from c("Importances", "EvalPlots", "EvalMetrics", "PDFs", "Score_TrainData")

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)

PrimaryDateColumn

Supply a date or datetime column for model evaluation plots

WeightsColumnName

Supply a column name for your weights column. Leave NULL otherwise

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.

DebugMode Set to TRUE to get a print out of the steps taken throughout the function

SaveInfoToPDF Set to TRUE to save model insights to pdf

ModelID A character string to name your model and output

EncodingMethod Choose from 'binary', 'm_estimator', 'credibility', 'woe', 'target_encoding', 'poly_encode', 'backward_difference', 'helmert'

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", "Sqrt", "Logit", "YeoJohnson". 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"

grid_eval_metric

        "mae", "mape", "rmse", "r2". Case sensitive

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.

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

        Runs without new winner to end procedure

MaxRunMinutes In minutes

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

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(), AutoLightGBMRegression(), AutoNLS()

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 = parallel::detectCores(),
  LossFunction = 'reg:squarederror',

  # Metadata args
  OutputSelection = c('Importances', 'EvalPlots', 'EvalMetrics', 'Score_TrainData'),
  model_path = normalizePath("./"),
  metadata_path = NULL,
  ModelID = "Test_Model_1",
  EncodingMethod = 'credibility',
  ReturnFactorLevels = TRUE,
  ReturnModelObjects = TRUE,
  SaveModelObjects = FALSE,
  SaveInfoToPDF = FALSE,
  DebugMode = FALSE,

  # Data args
  data = data,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = 'Adrian',
  FeatureColNames = names(data)[!names(data) %in%
    c('IDcol_1', 'IDcol_2', 'Adrian')],
  PrimaryDateColumn = NULL,
  WeightsColumnName = NULL,
  IDcols = c('IDcol_1', 'IDcol_2'),
  TransformNumericColumns = NULL,
  Methods = c('Asinh', 'Asin', 'Log', 'LogPlus1', 'Sqrt', 'Logit'),
```

```
      # Model evaluation args
      eval_metric = 'rmse',
      NumOfParDepPlots = 3L,

      # Grid tuning args
      PassInGrid = NULL,
      GridTune = FALSE,
      grid_eval_metric = 'r2',
      BaselineComparison = 'default',
      MaxModelsInGrid = 10L,
      MaxRunsWithoutNewWinner = 20L,
      MaxRunMinutes = 24L*60L,
      Verbose = 1L,

      # ML args
      Trees = 50L,
      eta = 0.05,
      max_depth = 4L,
      min_child_weight = 1.0,
      subsample = 0.55,
      colsample_bytree = 0.55)

  ## End(Not run)
```

AutoXGBoostScoring   *AutoXGBoostScoring*

Description

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

Usage

```
AutoXGBoostScoring(
  TargetType = NULL,
  ScoringData = NULL,
  ReturnShapValues = FALSE,
  FeatureColumnNames = NULL,
  IDcols = NULL,
  EncodingMethod = "binary",
  FactorLevelsList = NULL,
  TargetLevels = NULL,
  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 AutoXGBoostRegression(), AutoXGBoostClassify() or AutoXGBoostMultiClass()
ScoringData	This is your data.table of features for scoring. Can be a single row or batch.
ReturnShapValues	
	Set to TRUE to return shap values for the predicted values
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
EncodingMethod	Choose from 'binary', 'm_estimator', 'credibility', 'woe', 'target_encoding', 'poly_encode', 'backward_difference', 'helmert'
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.
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()`, `AutoLightGBMScoring()`

Examples

```
## Not run:
Preds <- AutoXGBoostScoring(
  TargetType = "regression",
  ScoringData = data,
  ReturnShapValues = FALSE,
  FeatureColumnNames = 2:12,
  IDcols = NULL,
  EncodingMethod = "binary",
  FactorLevelsList = NULL,
  TargetLevels = NULL,
  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)
```

Bisection	*Bisection*

Description

Finds roots for a given interval of values for a given function using bisection method
Algorithms

Usage

```
Bisection(f = function(x) x^2 - 4 * x + 3, a = 0, b = 2)
```

Arguments

f	mathematical function
a	lower bound numeric value
b	upper bound numeric value

Author(s)

Adrian Antico

Examples

```
## Not run:
RemixAutoML::Bisection(f = function(x) x ^ 2 - 4 * x + 3, a = 0, b = 2)
# 1

## End(Not run)
```

BlankRow	*BlankRow*

Description

BlankRow add blank row with width w

Usage

```
BlankRow(W)
```

Arguments

W	width of column

Value

Adds a row to your UI of width W

Author(s)

Adrian Antico

See Also

Other Shiny: `ArgNullCheck2()`, `ArgNullCheck()`, `DateInput()`, `GenerateEvaluationMetrics()`, `NumericInput()`, `PickerInput_GetLevels()`, `PickerInput()`, `PreparePlotData()`, `ReactiveLoadCSV()`, `ReturnParam()`, `StoreArgs()`, `TextInput()`, `observeEventLoad()`

Examples

```
## Not run:
RemixAutoML::BlankRow(12)

## End(Not run)
```

CategoricalEncoding   *CategoricalEncoding*

Description

Categorical encoding for factor and character columns

Usage

```
CategoricalEncoding(
  data = NULL,
  ML_Type = "classification",
  GroupVariables = NULL,
  TargetVariable = NULL,
  Method = NULL,
  SavePath = NULL,
  Scoring = FALSE,
  ImputeValueScoring = NULL,
  ReturnFactorLevelList = TRUE,
  SupplyFactorLevelList = NULL,
  KeepOriginalFactors = TRUE
)
```

Arguments

data	Source data.table
ML_Type	Only use with Method "credibility". Select from 'classification' or 'regression'.
GroupVariables	Columns to encode
Method	Method to utilize. Choose from 'm_estimator', 'credibility', 'woe', 'target_encoding', 'poly_encode', 'backward_difference', 'helmert'

SavePath   Path to save artifacts for recreating in scoring environments

Scoring   Set to TRUE for scoring mode.

ImputeValueScoring

                  If levels cannot be matched on scoring data you can supply a value to impute the NA's. Otherwise, leave NULL and manage them outside the function

ReturnFactorLevelList

                  TRUE by default. Returns a list of the factor variable and transformations needed for regenerating them in a scoring environment. Alternatively, if you save them to file, they can be called for use in a scoring environment.

SupplyFactorLevelList

                  The FactorCompenents list that gets returned. Supply this to recreate features in scoring environment

KeepOriginalFactors

                  Defaults to TRUE. Set to FALSE to remove the original factor columns

TargetVariabl   Target column name

Author(s)

Adrian Antico

See Also

Other Feature Engineering: `AutoDataPartition()`, `AutoDiffLagN()`, `AutoHierarchicalFourier()`, `AutoInteraction()`, `AutoLagRollMode()`, `AutoLagRollStatsScoring()`, `AutoLagRollStats()`, `AutoTransformationCreate()`, `AutoTransformationScore()`, `AutoWord2VecModeler()`, `AutoWord2VecScoring()`, `CreateCalendarVariables()`, `CreateHolidayVariables()`, `DummifyDT()`, `H2OAutoencoderScoring()`, `H2OAutoencoder()`, `ModelDataPrep()`, `TimeSeriesFill()`

Examples

```
## Not run:
# Create fake data with 10 categorical
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000000,
  ID = 2L,
  ZIP = 0,
  FactorCount = 10L,
  AddDate = FALSE,
  Classification = TRUE,
  MultiClass = FALSE)

# Take your pick
Meth <- c('m_estimator',
          'credibility',
          'woe',
          'target_encoding',
          'poly_encode',
          'backward_difference',
          'helmert')

# Pass to function
MethNum <- 1
```

```
# Mock test data with same factor levels
test <- data.table::copy(data)

# Run in Train Mode
data <- RemixAutoML::CategoricalEncoding(
  data = data,
  ML_Type = "classification",
  GroupVariables = paste0("Factor_", 1:10),
  TargetVariable = "Adrian",
  Method = Meth[MethNum],
  SavePath = getwd(),
  Scoring = FALSE,
  ReturnFactorLevelList = FALSE,
  SupplyFactorLevelList = NULL,
  KeepOriginalFactors = FALSE)

# View results
print(data)

# Run in Score Mode by pulling in the csv's
test <- RemixAutoML::CategoricalEncoding(
  data = data,
  ML_Type = "classification",
  GroupVariables = paste0("Factor_", 1:10),
  TargetVariable = "Adrian",
  Method = Meth[MethNum],
  SavePath = getwd(),
  Scoring = TRUE,
  ImputeValueScoring = 222,
  ReturnFactorLevelList = FALSE,
  SupplyFactorLevelList = NULL,
  KeepOriginalFactors = FALSE)

## End(Not run)
```

CharNull   *CharNull*

Description

CharNull

Usage

```
CharNull(x)
```

Arguments

x   Value

ChartTheme *ChartTheme*

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 Graphics: 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 + ChartTheme(Size = 12)

## End(Not run)
```

```
CreateCalendarVariables
```
CreateCalendarVariables

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(), AutoDiffLagN(), AutoHierarchicalFourier(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring() CategoricalEncoding(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep(), 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

Description

CreateHolidayVariables Rapidly creates holiday count variables based on the date columns you provide

Usage

```
CreateHolidayVariables(
  data,
  DateCols = NULL,
  LookbackDays = NULL,
  HolidayGroups = c("USPublicHolidays", "EasterGroup", "ChristmasGroup",
    "OtherEcclesticalFeasts"),
  Holidays = 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
LookbackDays	Default NULL which investigates Date - Lag1Date to compute Holiday's per period. Otherwise it will lookback LokkbackDays.
HolidayGroups	Pick groups
Holidays	Pick holidays
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()`, `AutoDiffLagN()`, `AutoHierarchicalFourier()`, `AutoInteraction()`, `AutoLagRollMode()`, `AutoLagRollStatsScoring()`, `AutoLagRollStats()`, `AutoTransformationCreate()`, `AutoTransformationScore()`, `AutoWord2VecModeler()`, `AutoWord2VecScoring()`, `CategoricalEncoding()`, `CreateCalendarVariables()`, `DummifyDT()`, `H2OAutoencoderScoring()`, `H2OAutoencoder()`, `ModelDataPrep()`, `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",
    LookbackDays = NULL,
    HolidayGroups = c("USPublicHolidays","EasterGroup",
      "ChristmasGroup","OtherEcclesticalFeasts"),
    Holidays = NULL,
    Print = FALSE))
head(data)
print(runtime)

## End(Not run)
```

CumGainsChart *CumGainsChart*

Description

Create a cumulative gains chart

Usage

```
CumGainsChart(
  data = NULL,
  PredictedColumnName = "p1",
  TargetColumnName = NULL,
  NumBins = 20,
```

```
  SavePlot = FALSE,
  Name = NULL,
  metapath = NULL,
  modelpath = NULL
)
```

Arguments

data	Test data with predictions. data.table
PredictedColumnName	
	Name of column that is the model score
TargetColumnName	
	Name of your target variable column
NumBins	Number of percentile bins to plot
SavePlot	FALSE by default
Name	File name for saving
metapath	Path to directory
modelpath	Path to directory

Author(s)

Adrian Antico

See Also

Other Model Evaluation and Interpretation: AutoShapeShap(), EvalPlot(), ParDepCalPlots(), ROCPlot(), RedYellowGreen(), ResidualPlots(), SingleRowShapeShap(), threshOptim()

DateInput   *DateInput*

Description

DateInput automatically builds a date input with ProjectList argument usage if it exists

Usage

```
DateInput(
  InputID = "TS_CARMA_HolidayMovingAverages",
  Label = "Import Data Creation Date",
  Value = Sys.Date(),
  Min = "1970-01-01",
  Max = "2100-01-01",
  Format = "yyyy-mm-dd"
)
```

Arguments

InputID	Feeds ProjectList and inputId. Argument saved in ProjectList
Label	Feeds label
Value	Default
Min	Min value
Max	Max value
Format	Date format

Value

PickerInput object for server.R to go into renderUI(PickerInput())

Author(s)

Adrian Antico

See Also

Other Shiny: `ArgNullCheck2()`, `ArgNullCheck()`, `BlankRow()`, `GenerateEvaluationMetrics()`, `NumericInput()`, `PickerInput_GetLevels()`, `PickerInput()`, `PreparePlotData()`, `ReactiveLoadCSV()`, `ReturnParam()`, `StoreArgs()`, `TextInput()`, `observeEventLoad()`

Examples

```
## Not run:
output$TS_Date <- renderUI({
  RemixAutoML::DateInput(InputID = "TS_CARMA_HolidayMovingAverages",
                         Label = "Import Data Creation Date",
                         Value = Sys.Date(),
                         Min = "1970-01-01",
                         Max = "2100-01-01",
                         Format = "yyyy-mm-dd")})

## End(Not run)
```

DummifyDT   *DummifyDT*

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,
  TopN = NULL,
  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
TopN	Default is NULL. Scalar to apply to all categorical columns or a vector to apply to each categorical variable. Only create dummy variables for the TopN number of levels. Will be either TopN or max(levels)
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. It converts the added dummy column names to conform with H2O dummy variable naming convention
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()`, `AutoDiffLagN()`, `AutoHierarchicalFourier()`, `AutoInteraction()`, `AutoLagRollMode()`, `AutoLagRollStatsScoring()`, `AutoLagRollStats()`, `AutoTransformationCreate()`, `AutoTransformationScore()`, `AutoWord2VecModeler()`, `AutoWord2VecScoring()`, `CategoricalEncoding()`, `CreateCalendarVariables()`, `CreateHolidayVariables()`, `H2OAutoencoderScoring()`, `H2OAutoencoder()`, `ModelDataPrep()`, `TimeSeriesFill()`

Examples

```
## Not run:
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"),
  TopN = c(rep(3,9)),
  KeepFactorCols = TRUE,
  OneHot = FALSE,
  SaveFactorLevels = TRUE,
  SavePath = getwd(),
  ImportFactorLevels = FALSE,
  FactorLevelsList = NULL,
  ClustScore = FALSE,
  ReturnFactorLevels = FALSE)

# Create Fake Data for Scoring Replication
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.85,
  N = 25000,
  ID = 2L,
  ZIP = 0,
  FactorCount = 10L,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Scoring Version
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"),
  TopN = c(rep(3,9)),
  KeepFactorCols = TRUE,
  OneHot = FALSE,
  SaveFactorLevels = TRUE,
  SavePath = getwd(),
  ImportFactorLevels = TRUE,
  FactorLevelsList = NULL,
  ClustScore = FALSE,
  ReturnFactorLevels = FALSE)

## End(Not run)
```

EDA_Histograms   *EDA_Histograms*

Description

Creates histograms

Usage

```
EDA_Histograms(
  data = NULL,
  PlotColumns = NULL,
  SampleCount = 1e+05,
  SavePath = NULL,
  FactorCountPerPlot = 10,
  AddDensityLine = FALSE,
  PrintOutput = FALSE,
  Size = 12,
  AngleX = 35,
  AngleY = 0,
  ChartColor = "lightsteelblue1",
  BorderColor = "darkblue",
  TextColor = "darkblue",
  GridColor = "white",
  BackGroundColor = "gray95",
  LegendPosition = "bottom"
)
```

Arguments

data	Input data.table
PlotColumns	Default NULL. If NULL, all columns will be plotted (except date cols). Otherwise, supply a character vector of columns names to plot
SampleCount	Number of random samples to use from data. data is first shuffled and then random samples taken
SavePath	Output file path to where you can optionally save pdf
FactorCountPerPlot	
	Default 10

AddDensityLine	Set to TRUE to add a density line to the plots
PrintOutput	Default FALSE. TRUE will print results upon running function
Size	Default 12
AngleX	Default 35
AngleY	Default 0
ChartColor	Default "lightsteelblue1"
BorderColor	Default "darkblue"
TextColor	Default "darkblue"
GridColor	Default "white"
BackGroundColor	
	Default "gray95"
LegendPosition	Default "bottom"

Author(s)

Adrian Antico

See Also

Other EDA: [AutoWordFreq](), [Mode](), [PlotGUI](), [ScatterCopula](), [UserBaseEvolution]()

EvalPlot	*EvalPlot*

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: AutoShapeShap(), CumGainsChart(), ParDepCalPlots(), ROCPlot(), RedYellowGreen(), ResidualPlots(), SingleRowShapeShap(), 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
RemixAutoML::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

Create fake data for examples

Usage

```
FakeDataGenerator(
  Correlation = 0.7,
  N = 1000L,
  ID = 5L,
  FactorCount = 2L,
  AddDate = TRUE,
  AddComment = FALSE,
  AddWeightsColumn = FALSE,
  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
`AddComment`	Set to TRUE to add a comment 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

Examples

```
## Not run:
data <- RemixAutoML::FakeDataGenerator(
   Correlation = 0.70,
   N = 1000L,
   ID = 2L,
   FactorCount = 2L,
   AddDate = TRUE,
   AddWeightsColumn = FALSE,
   AddComment = FALSE,
   ZIP = 2L,
   TimeSeries = FALSE,
   ChainLadderData = FALSE,
   Classification = FALSE,
   MultiClass = FALSE)

## End(Not run)
```

FilterLogicData *FilterLogicData*

Description

FilterLogicData

Usage

```
FilterLogicData(
  data1,
  input,
  FilterLogic = input[["FilterLogic"]],
  FilterVariable = input[["FilterVariable_1"]],
  FilterValue = input[["FilterValue_1a"]],
  FilterValue2 = input[["FilterValue_1b"]],
  Debug = FALSE
)
```

Arguments

data1	data.table
input	passthrough
FilterLogic	passthrough
FilterVariable	passthrough
FilterValue	passthrough
FilterValue2	passthrough

FilterValues *FilterValues*

Description

FilterValues

Usage

```
FilterValues(data, VarName = input[["FilterVariable_1"]], type = 1)
```

Arguments

data	data.table
VarName	Variable name
type	1 for min, 2 for max

GenerateEvaluationMetrics

GenerateEvaluationMetrics

Description

GenerateEvaluationMetrics calculates evaluation metrics for out of sample forecast and evaluation data

Usage

```
GenerateEvaluationMetrics(
  EvalData = NULL,
  TargetName = NULL,
  DateName = NULL,
  GroupNames = NULL
)
```

Arguments

EvalData	Source data in shiny app
TargetName	Target variable name
DateName	Date variable name
GroupNames	Group variable names

Value

PreparePlotData object for server.R to

Author(s)

Adrian Antico

See Also

Other Shiny: ArgNullCheck2(), ArgNullCheck(), BlankRow(), DateInput(), NumericInput(),
PickerInput_GetLevels(), PickerInput(), PreparePlotData(), ReactiveLoadCSV(), ReturnParam(),
StoreArgs(), TextInput(), observeEventLoad()

Examples

```
## Not run:
 PlotData <- RemixAutoML::PreparePlotData(input, TargetVariable = "TargetVariables", DateVariable = "DateVari

## End(Not run)
```

GenTSAnomVars   *GenTSAnomVars*

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: AutoClusteringScoring(), AutoClustering(), H2OIsolationForestScoring(), 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)
```

GetFilterValueLabel   *GetFilterValueLabel*

Description

GetFilterValueLabel

Usage

```
GetFilterValueLabel(data, VarName = NULL, type = 1)
```

Arguments

data	data.table
VarName	Variable name
type	1 for min, 2 for max

GetFilterValueMultiple

GetFilterValueMultiple

Description

GetFilterValueMultiple

Usage

```
GetFilterValueMultiple(data, VarName = NULL, type = 1)
```

Arguments

data	data.table
VarName	Variable name
type	1 for min, 2 for max

H2OAutoencoder   *H2OAutoencoder*

Description

H2OAutoencoder for anomaly detection and or dimensionality reduction

Usage

```
H2OAutoencoder(
  AnomalyDetection = FALSE,
  DimensionReduction = TRUE,
  data,
  Features = NULL,
  RemoveFeatures = FALSE,
  NThreads = max(1L, parallel::detectCores() - 2L),
  MaxMem = "28G",
  H2OStart = TRUE,
  H2OShutdown = TRUE,
  ModelID = "TestModel",
  model_path = NULL,
  LayerStructure = NULL,
  NodeShrinkRate = (sqrt(5) - 1)/2,
  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
`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"
`H2OStart`	TRUE to start H2O inside the function
`H2OShutdown`	Setting to TRUE will shutdown H2O when it done being used internally.
`ModelID`	"TestModel"
`model_path`	If NULL no model will be saved. If a valid path is supplied the model will be saved there
`LayerStructure`	If NULL, layers and sizes will be created for you, using NodeShrinkRate and 7 layers will be created.
`NodeShrinkRate`	= (sqrt(5) - 1) / 2,
`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()`](), [`AutoDiffLagN()`](), [`AutoHierarchicalFourier()`](), [`AutoInteraction()`](), [`AutoLagRollMode()`](), [`AutoLagRollStatsScoring()`](), [`AutoLagRollStats()`](), [`AutoTransformationCreate()`](), [`AutoTransformationScore()`](), [`AutoWord2VecModeler()`](), [`AutoWord2VecScoring()`](), [`CategoricalEncoding()`](), [`CreateCalendarVariables()`](), [`CreateHolidayVariables()`](), [`DummifyDT()`](), [`H2OAutoencoderScoring()`](), [`ModelDataPrep()`](), [`TimeSeriesFill()`]()

Examples

```
## Not run:
############################
# Training
############################

# Create simulated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.70,
  N = 1000L,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  AddComment = FALSE,
  ZIP = 2L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run algo
Output <- RemixAutoML::H2OAutoencoder(

  # Select the service
  AnomalyDetection = TRUE,
  DimensionReduction = TRUE,

  # Data related args
  data = data,
  Features = names(data)[2L:(ncol(data)-1L)],
  per_feature = FALSE,
  RemoveFeatures = FALSE,
  ModelID = "TestModel",
  model_path = getwd(),

  # H2O Environment
  NThreads = max(1L, parallel::detectCores()-2L),
  MaxMem = "28G",
  H2OStart = TRUE,
  H2OShutdown = TRUE,

  # H2O ML Args
  LayerStructure = NULL,
  NodeShrinkRate = (sqrt(5) - 1) / 2,
  ReturnLayer = 4L,
  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

############################
# Scoring
############################

# Create simulated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.70,
  N = 1000L,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  AddComment = FALSE,
  ZIP = 2L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run algo
data <- RemixAutoML::H2OAutoencoderScoring(

  # Select the service
  AnomalyDetection = TRUE,
  DimensionReduction = TRUE,

  # Data related args
  data = data,
  Features = names(data)[2L:ncol(data)],
  RemoveFeatures = TRUE,
  per_feature = FALSE,
  ModelObject = NULL,
  ModelID = "TestModel",
  model_path = getwd(),

  # H2O args
  NThreads = max(1L, parallel::detectCores()-2L),
  MaxMem = "28G",
  H2OStart = TRUE,
  H2OShutdown = TRUE,
  ReturnLayer = 4L)

## End(Not run)
```

H2OAutoencoderScoring   *H2OAutoencoderScoring*

Description

H2OAutoencoderScoring for anomaly detection and or dimensionality reduction

Usage

```
H2OAutoencoderScoring(
  data,
  Features = NULL,
  RemoveFeatures = FALSE,
  ModelObject = NULL,
  AnomalyDetection = TRUE,
  DimensionReduction = TRUE,
  ReturnLayer = 4L,
  per_feature = TRUE,
  NThreads = max(1L, parallel::detectCores() - 2L),
  MaxMem = "28G",
  H2OStart = TRUE,
  H2OShutdown = TRUE,
  ModelID = "TestModel",
  model_path = NULL
)
```

Arguments

`data`	The data.table with the columns you wish to have analyzed
`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
`ModelObject`	If NULL then the model will be loaded from file. Otherwise, it will use what is supplied
`AnomalyDetection`	
	Set to TRUE to run anomaly detection
`DimensionReduction`	
	Set to TRUE to run dimension reduction
`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
`NThreads`	max(1L, parallel::detectCores()-2L)
`MaxMem`	"28G"
`H2OStart`	TRUE to start H2O inside the function
`H2OShutdown`	Setting to TRUE will shutdown H2O when it done being used internally.
`ModelID`	"TestModel"
`model_path`	If NULL no model will be saved. If a valid path is supplied the model will be saved there

Value

A data.table

Author(s)

Adrian Antico

See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoHierarchicalFourier(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring() CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoder(), ModelDataPrep(), TimeSeriesFill()

Examples

```
## Not run:
############################
# Training
############################

# Create simulated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.70,
  N = 1000L,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  AddComment = FALSE,
  ZIP = 2L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run algo
data <- RemixAutoML::H2OAutoencoder(

  # Select the service
  AnomalyDetection = TRUE,
  DimensionReduction = TRUE,

  # Data related args
  data = data,
  ValidationData = NULL,
  Features = names(data)[2L:(ncol(data)-1L)],
  per_feature = FALSE,
  RemoveFeatures = TRUE,
  ModelID = "TestModel",
  model_path = getwd(),

  # H2O Environment
  NThreads = max(1L, parallel::detectCores()-2L),
  MaxMem = "28G",
  H2OStart = TRUE,
  H2OShutdown = TRUE,

  # H2O ML Args
  LayerStructure = NULL,
  ReturnLayer = 4L,
  Activation = "Tanh",
  Epochs = 5L,
  L2 = 0.10,
```

```
  ElasticAveraging = TRUE,
  ElasticAveragingMovingRate = 0.90,
  ElasticAveragingRegularization = 0.001)

#############################
# Scoring
#############################

# Create simulated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.70,
  N = 1000L,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  AddComment = FALSE,
  ZIP = 2L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run algo
data <- RemixAutoML::H2OAutoencoderScoring(

  # Select the service
  AnomalyDetection = TRUE,
  DimensionReduction = TRUE,

  # Data related args
  data = data,
  Features = names(data)[2L:ncol(data)],
  RemoveFeatures = TRUE,
  per_feature = FALSE,
  ModelObject = NULL,
  ModelID = "TestModel",
  model_path = getwd(),

  # H2O args
  NThreads = max(1L, parallel::detectCores()-2L),
  MaxMem = "28G",
  H2OStart = TRUE,
  H2OShutdown = TRUE,
  ReturnLayer = 4L)

## End(Not run)
```

H2OIsolationForest   *H2OIsolationForest*

Description

H2OIsolationForestScoring for dimensionality reduction and / or anomaly detection

Usage

```
H2OIsolationForest(
  data,
  Features = NULL,
  IDcols = NULL,
  ModelID = "TestModel",
  SavePath = NULL,
  Threshold = 0.975,
  MaxMem = "28G",
  NThreads = -1,
  NTrees = 100,
  MaxDepth = 8,
  MinRows = 1,
  RowSampleRate = (sqrt(5) - 1)/2,
  ColSampleRate = 1,
  ColSampleRatePerLevel = 1,
  ColSampleRatePerTree = 1,
  CategoricalEncoding = c("AUTO"),
  Debug = FALSE
)
```

Arguments

data	The data.table with the columns you wish to have analyzed
Features	A character vector with the column names to utilize in the isolation forest
IDcols	A character vector with the column names to not utilize in the isolation forest but have returned with the data output. Otherwise those columns will be removed
ModelID	Name for model that gets saved to file if SavePath is supplied and valid
SavePath	Path directory to store saved model
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
MaxDepth	Max tree depth
MinRows	Minimum number of rows allowed per leaf
RowSampleRate	Number of rows to sample per tree
ColSampleRate	Sample rate for each split
ColSampleRatePerLevel	
	Sample rate for each level
ColSampleRatePerTree	
	Sample rate per tree
CategoricalEncoding	
	Choose from "AUTO", "Enum", "OneHotInternal", "OneHotExplicit", "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited"
Debug	Debugging

Value

Source data.table with predictions. Note that any columns not listed in Features nor IDcols will not be returned with data. If you want columns returned but not modeled, supply them as IDcols

Author(s)

Adrian Antico

See Also

Other Unsupervised Learning: AutoClusteringScoring(), AutoClustering(), GenTSAnomVars(), H2OIsolationForestScoring(), ResidualOutliers()

Examples

```
## Not run:
# Create simulated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.70,
  N = 50000,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  ZIP = 0L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run algo
data <- RemixAutoML::H2OIsolationForest(
  data,
  Features = names(data)[2L:ncol(data)],
  IDcols = c("Adrian", "IDcol_1", "IDcol_2"),
  ModelID = "Adrian",
  SavePath = getwd(),
  Threshold = 0.95,
  MaxMem = "28G",
  NThreads = -1,
  NTrees = 100,
  MaxDepth = 8,
  MinRows = 1,
  RowSampleRate = (sqrt(5)-1)/2,
  ColSampleRate = 1,
  ColSampleRatePerLevel = 1,
  ColSampleRatePerTree = 1,
  CategoricalEncoding = c("AUTO"),
  Debug = TRUE)

# Remove output from data and then score
data[, eval(names(data)[17:ncol(data)]) := NULL]

# Run algo
Outliers <- RemixAutoML::H2OIsolationForestScoring(
  data,
  Features = names(data)[2:ncol(data)],
```

```
    IDcols = c("Adrian", "IDcol_1", "IDcol_2"),
    H2OStart = TRUE,
    H2OShutdown = TRUE,
    ModelID = "TestModel",
    SavePath = getwd(),
    Threshold = 0.95,
    MaxMem = "28G",
    NThreads = -1,
    Debug = FALSE)

## End(Not run)
```

H2OIsolationForestScoring

H2OIsolationForestScoring

Description

H2OIsolationForestScoring for dimensionality reduction and / or anomaly detection scoring on new data

Usage

```
H2OIsolationForestScoring(
  data,
  Features = NULL,
  IDcols = NULL,
  H2OStart = TRUE,
  H2OShutdown = TRUE,
  ModelID = "TestModel",
  SavePath = NULL,
  Threshold = 0.975,
  MaxMem = "28G",
  NThreads = -1,
  Debug = FALSE
)
```

Arguments

data	The data.table with the columns you wish to have analyzed
Features	A character vector with the column names to utilize in the isolation forest
IDcols	A character vector with the column names to not utilize in the isolation forest but have returned with the data output. Otherwise those columns will be removed
H2OStart	TRUE to have H2O started inside function
H2OShutdown	TRUE to shutdown H2O inside function
ModelID	Name for model that gets saved to file if SavePath is supplied and valid
SavePath	Path directory to store saved model
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)
Debug	Debugging

Value

Source data.table with predictions. Note that any columns not listed in Features nor IDcols will not be returned with data. If you want columns returned but not modeled, supply them as IDcols

Author(s)

Adrian Antico

See Also

Other Unsupervised Learning: AutoClusteringScoring(), AutoClustering(), GenTSAnomVars(), H2OIsolationForest(), ResidualOutliers()

Examples

```
## Not run:
# Create simulated data
data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.70,
  N = 50000,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  ZIP = 0L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run algo
data <- RemixAutoML::H2OIsolationForest(
  data,
  Features = names(data)[2L:ncol(data)],
  IDcols = c("Adrian", "IDcol_1", "IDcol_2"),
  ModelID = "Adrian",
  SavePath = getwd(),
  Threshold = 0.95,
  MaxMem = "28G",
  NThreads = -1,
  NTrees = 100,
  SampleRate = (sqrt(5)-1)/2,
  MaxDepth = 8,
  MinRows = 1,
  ColSampleRate = 1,
  ColSampleRatePerLevel = 1,
  ColSampleRatePerTree = 1,
  CategoricalEncoding = c("AUTO"),
  Debug = TRUE)

# Remove output from data and then score
data[, eval(names(data)[17:ncol(data)]) := NULL]

# Run algo
Outliers <- RemixAutoML::H2OIsolationForestScoring(
  data,
  Features = names(data)[2:ncol(data)],
```

```
   IDcols = c("Adrian", "IDcol_1", "IDcol_2"),
   H2OStart = TRUE,
   H2OShutdown = TRUE,
   ModelID = "TestModel",
   SavePath = getwd(),
   Threshold = 0.95,
   MaxMem = "28G",
   NThreads = -1,
   Debug = FALSE)

## End(Not run)
```

InsertSortedValue   *InsertSortedValue*

Description

Update a sorted vector with a new value that preserves sort order

Algorithms

Usage

```
InsertSortedValue(Vec, Val, order = "left")
```

Arguments

Val	value to insert
order	'left' or 'right', insert value location
vec	numeric vector

Author(s)

Adrian Antico

Examples

```
## Not run:
RemixAutoML::InsertSortedValue(vec = seq(2, 2000, 2), Val = 741, order = "left")

## End(Not run)
```

IntNull   *IntNull*

Description

IntNull

Usage

```
IntNull(x)
```

Arguments

x   value

KeyVarsInit   *KeyVarsInit*

Description

KeyVarsInit

Usage

```
KeyVarsInit(data, VarName = NULL, type = 1)
```

Arguments

data	data.table
VarName	Variable name
type	1 for min, 2 for max

Mode   *Mode*

Description

Statistical mode. Only returns the first mode if there are many

Usage

```
Mode(x)
```

Arguments

x   vector

Author(s)

Adrian Antico

See Also

Other EDA: `AutoWordFreq()`, `EDA_Histograms()`, `PlotGUI()`, `ScatterCopula()`, `UserBaseEvolution()`

ModelDataPrep   *ModelDataPrep*

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,
  LogicalToBinary = FALSE,
  DateToChar = FALSE,
  IDateConversion = 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 LogicalToBinary	Defaults to TRUE which tells the function to convert integers to numeric
	Converts logical values to binary numeric values
DateToChar IDateConversion	Converts date columns into character columns
	Convert IDateTime to POSIXct and IDate to Date types
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(), AutoDiffLagN(), AutoHierarchicalFourier(),
AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(),
AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring()
CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(),
H2OAutoencoderScoring(), H2OAutoencoder(), 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,
  LogicalToBinary = FALSE,
  DateToChar   = FALSE,
  IDateConversion = FALSE,
  RemoveDates  = TRUE,
  MissFactor   = "0",
  MissNum      = -1,
  IgnoreCols   = c("Factor_1"))

# Check column types
str(data)

## End(Not run)
```

ModelInsightsReport *ModelInsightsReport*

Description

ModelInsightsReport is an Rmarkdown report for viewing the model insights generated by Remix-AutoML supervised learning functions

Usage

```
ModelInsightsReport(
  KeepOutput = NULL,
  TrainData = NULL,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  PredictionColumnName = "Predict",
  FeatureColumnNames = NULL,
  DateColumnName = NULL,
  TargetType = "regression",
  ModelID = "ModelTest",
  Algo = "catboost",
  SourcePath = NULL,
  OutputPath = NULL,
  RemixOutput = NULL,
  Test_Importance_dt = NULL,
  Validation_Importance_dt = NULL,
  Train_Importance_dt = NULL,
  Test_Interaction_dt = NULL,
  Validation_Interaction_dt = NULL,
  Train_Interaction_dt = NULL,
  GlobalVars = ls()
)
```

Arguments

KeepOutput	NULL A list of output names to select. Pass in as a character vector. E.g. c('Test_VariableImportance', 'Train_VariableImportance')
TrainData	data.table or something that converts to data.table via as.data.table
ValidationData	data.table or something that converts to data.table via as.data.table
TestData	data.table or something that converts to data.table via as.data.table
TargetColumnName	
	NULL. Target variable column name as character
PredictionColumnName	
	NULL. Predicted value column name as character. 'p1' for RemixAutoML functions
FeatureColumnNames	
	NULL. Feature column names as character vector.
DateColumnName	NULL. Date column name as character

TargetType	'regression', 'classification', or 'multiclass'
ModelID	ModelID used in the RemixAutoML supervised learning function
Algo	'catboost' or 'other'. Use 'catboost' if using RemixAutoML::AutoCatBoost_() functions. Otherwise, 'other'
SourcePath	Path to directory with RemixAutoML Model Output
OutputPath	Path to directory where the html will be saved
RemixOutput	Returned output from regression, classificaiton, and multiclass Remix Auto_() models. Currenly supports CatBoost, XGBoost, and LightGBM models
Test_Importance_dt	NULL.. Ignore if using RemixAutoML Models. Otherwise, supply a two column data.table with colnames 'Variable' and 'Importance'
Validation_Importance_dt	NULL.. Ignore if using RemixAutoML Models. Otherwise, supply a two column data.table with colnames 'Variable' and 'Importance'
Train_Importance_dt	NULL.. Ignore if using RemixAutoML Models. Otherwise, supply a two column data.table with colnames 'Variable' and 'Importance'
Test_Interaction_dt	NULL.. Ignore if using RemixAutoML Models. Otherwise, supply a three column data.table with colnames 'Features1', 'Features2' and 'score'
Validation_Interaction_dt	NULL.. Ignore if using RemixAutoML Models. Otherwise, supply a three column data.table with colnames 'Features1', 'Features2' and 'score'
Train_Interaction_dt	NULL.. Ignore if using RemixAutoML Models. Otherwise, supply a three column data.table with colnames 'Features1', 'Features2' and 'score'
GlobalVars	ls() don't use
Path	Path to Model Output if RemixOutput is left NULL

Author(s)

Adrian Antico

Examples

```
## Not run:

#####################################################
# CatBoost
#####################################################

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

```r
# Copy data
data1 <- data.table::copy(data)

# Run function
RemixOutput <- RemixAutoML::AutoCatBoostRegression(

  # GPU or CPU and the number of available GPUs
  TrainOnFull = FALSE,
  task_type = 'GPU',
  NumGPUs = 1,
  DebugMode = FALSE,

  # Metadata args
  OutputSelection = c('Importances','EvalPlots','EvalMetrics','Score_TrainData'),
  ModelID = 'Test_Model_1',
  model_path = getwd(),
  metadata_path = getwd(),
  SaveModelObjects = FALSE,
  SaveInfoToPDF = FALSE,
  ReturnModelObjects = TRUE,

  # Data args
  data = data1,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = 'Adrian',
  FeatureColNames = names(data1)[!names(data1) %in% c('IDcol_1','IDcol_2','Adrian')],
  PrimaryDateColumn = NULL,
  WeightsColumnName = NULL,
  IDcols = c('IDcol_1','IDcol_2'),
  TransformNumericColumns = 'Adrian',
  Methods = c('Asinh','Asin','Log','LogPlus1','Sqrt','Logit'),

  # Model evaluation
  eval_metric = 'RMSE',
  eval_metric_value = 1.5,
  loss_function = 'RMSE',
  loss_function_value = 1.5,
  MetricPeriods = 10L,
  NumOfParDepPlots = ncol(data1)-1L-2L,

  # Grid tuning args
  PassInGrid = NULL,
  GridTune = FALSE,
  MaxModelsInGrid = 30L,
  MaxRunsWithoutNewWinner = 20L,
  MaxRunMinutes = 60*60,
  BaselineComparison = 'default',

  # ML args
  langevin = FALSE,
  diffusion_temperature = 10000,
  Trees = 500,
  Depth = 9,
  L2_Leaf_Reg = NULL,
  RandomStrength = 1,
  BorderCount = 128,
```

```
    LearningRate = NULL,
    RSM = 1,
    BootStrapType = NULL,
    GrowPolicy = 'SymmetricTree',
    model_size_reg = 0.5,
    feature_border_type = 'GreedyLogSum',
    sampling_unit = 'Object',
    subsample = NULL,
    score_function = 'Cosine',
    min_data_in_leaf = 1)

# Create Model Insights Report
RemixAutoML::ModelInsightsReport(

  # Items to keep in global environment when
  #   function finishes execution
  KeepOutput = 'Test_VariableImportance',

  # DataSets
  TrainData = NULL,
  ValidationData = NULL,
  TestData = NULL,

  # Meta info
  TargetColumnName = NULL,
  PredictionColumnName = NULL,
  FeatureColumnNames = NULL,
  DateColumnName = NULL,

  # Variable Importance
  Test_Importance_dt = NULL,
  Validation_Importance_dt = NULL,
  Train_Importance_dt = NULL,
  Test_Interaction_dt = NULL,
  Validation_Interaction_dt = NULL,
  Train_Interaction_dt = NULL,

  # Control options
  TargetType = 'regression',
  ModelID = 'ModelTest',
  Algo = 'catboost',
  SourcePath = getwd(),
  OutputPath = getwd(),
  RemixOutput = RemixOutput)

## End(Not run)
```

---

multiplot   *multiplot*

Description

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

Usage

```
multiplot(plotlist = NULL)
```

Arguments

plotlist   This is the list of your charts

Value

Multiple ggplots on a single image

Author(s)

Adrian Antico

See Also

Other Graphics: [ChartTheme]( )()

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))

## End(Not run)
```

NumericInput   *NumericInput*

Description

NumericInput automatically builds a numeric input with tryCatch's and ProjectList argument usage if it exists

Usage

```
NumericInput(
  InputID = "TS_CARMA_HolidayMovingAverages",
  Label = "Select Holiday Count MA's",
  Step = 10,
  Value = 1,
  Min = 1,
  Max = 10
)
```

Arguments

InputID	Feeds ProjectList and inputId. Argument saved in ProjectList
Label	Feeds label
Step	Feeds size in the options list
Value	Default
Min	Min value
Max	Max value

Value

PickerInput object for server.R to go into renderUI(PickerInput())

Author(s)

Adrian Antico

See Also

Other Shiny: `ArgNullCheck2()`, `ArgNullCheck()`, `BlankRow()`, `DateInput()`, `GenerateEvaluationMetrics()`, `PickerInput_GetLevels()`, `PickerInput()`, `PreparePlotData()`, `ReactiveLoadCSV()`, `ReturnParam()`, `StoreArgs()`, `TextInput()`, `observeEventLoad()`

Examples

```
## Not run:
output$TS_CARMA_HolidayMovingAverages <- renderUI({
  RemixAutoML::NumericInput(InputID = "TS_CARMA_HolidayMovingAverages",
                            Label = "Select Holiday Count MA's",
                            Min = 1,
                            Max = 1,
                            Value = 1,
```

```
                              Step = 1)})

## End(Not run)
```

NumNull   *NumNull*

Description

NumNull

Usage

```
NumNull(x)
```

Arguments

x   value

---

observeEventLoad   *observeEventLoad*

Description

Used to load .Rdata files in Shiny and assign the object a name

Usage

```
observeEventLoad(input, InputVal = NULL, ObjectName = NULL)
```

Arguments

input   Passthrough
InputVal   The values that goes after input$
ObjectName   The name of the object to assign the load output to

Author(s)

Adrian Antico

See Also

Other Shiny: ArgNullCheck2(), ArgNullCheck(), BlankRow(), DateInput(), GenerateEvaluationMetrics(), NumericInput(), PickerInput_GetLevels(), PickerInput(), PreparePlotData(), ReactiveLoadCSV(), ReturnParam(), StoreArgs(), TextInput()

ParDepCalPlots   *ParDepCalPlots*

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),
  DateColumn = NULL,
  DateAgg_3D = NULL,
  PlotYMeanColor = "black",
  PlotXMeanColor = "chocolate",
  PlotXLowColor = "purple",
  PlotXHighColor = "purple"
)
```

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.
DateColumn	Add date column for 3D scatterplot
DateAgg_3D	Aggregate date column by 'day', 'week', 'month', 'quarter', 'year'

Value

Partial dependence calibration plot or boxplot

Author(s)

Adrian Antico

See Also

Other Model Evaluation and Interpretation: AutoShapeShap(), CumGainsChart(), EvalPlot(), ROCPlot(), RedYellowGreen(), ResidualPlots(), SingleRowShapeShap(), 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),
  DateColumn = NULL,
  DateAgg_3D = NULL)

## End(Not run)
```

PickerInput   *PickerInput*

Description

PickerInput automatically builds a picker input with tryCatch's and ProjectList argument usage if it exists

Usage

```
PickerInput(
  InputID = "TS_CARMA_HolidayMovingAverages",
  Label = "Select Holiday Count MA's",
  Choices = as.character(0:50),
  SelectedDefault = as.character(c(1, 2)),
  Size = 10,
  SelectedText = "count > 1",
  Multiple = TRUE,
  ActionBox = TRUE,
  Debug = FALSE
)
```

Arguments

InputID	Feeds ProjectList and inputId. Argument saved in ProjectList
Label	Feeds label
Choices	Feeds choices
SelectedDefault	
	Feeds selected for cases where ProjectList has a null element
Size	Feeds size in the options list
SelectedText	Feeds selected-text-format in options list
Multiple	Feeds multiple for enabling selecting more than one element from list
ActionBox	Feeds actions-box for option list
Debug	FALSE

Value

PickerInput object for server.R to go into renderUI(PickerInput())

Author(s)

Adrian Antico

See Also

Other Shiny: `ArgNullCheck2()`, `ArgNullCheck()`, `BlankRow()`, `DateInput()`, `GenerateEvaluationMetrics()`,
`NumericInput()`, `PickerInput_GetLevels()`, `PreparePlotData()`, `ReactiveLoadCSV()`, `ReturnParam()`,
`StoreArgs()`, `TextInput()`, `observeEventLoad()`

Examples

```
## Not run:
output$TS_CARMA_HolidayMovingAverages <- renderUI({
 RemixAutoML::PickerInput(InputID = "TS_CARMA_HolidayMovingAverages", Label = "Select Holiday Count MA's", Ch
                 SelectedDefault = as.character(c(1,2)), Size = 10, SelectedText = "count > 1", Multiple = TRU

## End(Not run)
```

PickerInput_GetLevels   *PickerInput_GetLevels*

Description

PickerInput_GetLevels automatically builds a picker input with tryCatch's and ProjectList argument
usage if it exists

Usage

```
PickerInput_GetLevels(
  input,
  data = "SourceData",
  NumGroupVar = 3,
  InputID = "TS_CARMA_HolidayMovingAverages",
  InputID2 = "timeSeriesGroupVars",
  Choices = as.character(0:50),
  SelectedDefault = as.character(c(1, 2)),
  Size = 10,
  SelectedText = "count > 1",
  Multiple = TRUE,
  ActionBox = TRUE
)
```

Arguments

input	input object within shiny context
data	'SourceData' or whatever the name of your data is
NumGroupVar	Which group var to select
InputID	Feeds ProjectList and inputId. Argument saved in ProjectList
InputID2	Secondary object name
Choices	Feeds choices
SelectedDefault	
	Feeds selected for cases where ProjectList has a null element
Size	Feeds size in the options list
SelectedText	Feeds selected-text-format in options list
Multiple	Feeds multiple for enabling selecting more than one element from list
ActionBox	Feeds actions-box for option list

Value

PickerInput object for server.R to go into renderUI(PickerInput())

Author(s)

Adrian Antico

See Also

Other Shiny: `ArgNullCheck2()`, `ArgNullCheck()`, `BlankRow()`, `DateInput()`, `GenerateEvaluationMetrics()`, `NumericInput()`, `PickerInput()`, `PreparePlotData()`, `ReactiveLoadCSV()`, `ReturnParam()`, `StoreArgs()`, `TextInput()`, `observeEventLoad()`

Examples

```
## Not run:
output$TS_CARMA_HolidayMovingAverages <- renderUI({
  RemixAutoML::PickerInput_GetLevels(
   input, InputID = "TS_CARMA_HolidayMovingAverages", Label = "Select Holiday Count MA's", Choices = as.charact
   SelectedDefault = as.character(c(1,2)), Size = 10, SelectedText = "count > 1", Multiple = TRUE, ActionBox = T
```

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

PlotGUI   *PlotGUI*

Description

Spin up the esquisse plotting gui

Usage

```
PlotGUI()
```

See Also

Other EDA: AutoWordFreq(), EDA_Histograms(), Mode(), ScatterCopula(), UserBaseEvolution()

PreparePlotData   *PreparePlotData*

Description

PreparePlotData automatically builds a picker input with tryCatch's and ProjectList argument usage if it exists

Usage

```
PreparePlotData(
  input,
  PlotDataForecast,
  SubsetOnly = FALSE,
  Aggregate = NULL,
  TargetVariable = NULL,
  DateVariable = NULL,
  GroupVariables = NULL,
  G1Levels = NULL,
  G2Levels = NULL,
  G3Levels = NULL,
  Debug = FALSE
)
```

Arguments

input	input object within shiny context
PlotDataForecast	
	Source data in shiny app
SubsetOnly	Set to TRUE to only subset data
Aggregate	Session object indicating whether to use mean or sum

TargetVariable	Target variable name
DateVariable	Date variable name
GroupVariables	Group variable names
G1Levels	Name of group 1 levels list element
G2Levels	Name of group 2 levels list element
G3Levels	Name of group 3 levels list element
Debug	FALSE

Value

PreparePlotData object for server.R to

Author(s)

Adrian Antico

See Also

Other Shiny: ArgNullCheck2(), ArgNullCheck(), BlankRow(), DateInput(), GenerateEvaluationMetrics(), NumericInput(), PickerInput_GetLevels(), PickerInput(), ReactiveLoadCSV(), ReturnParam(), StoreArgs(), TextInput(), observeEventLoad()

Examples

```
## Not run:
  PlotData <- RemixAutoML::PreparePlotData(
    input,
    PlotDataForecast,
    SubsetOnly = FALSE,
    Aggregate = "mean",
    TargetVariable = "TargetVariables",
    DateVariable = "DateVariables",
    GroupVariables = GroupVariables,
    G1Levels = "TS_Group1Levels",
    G2Levels = "TS_Group2Levels",
    G3Levels = "TS_Group3Levels")

## End(Not run)
```

PrintToPDF *PrintToPDF*

Description

PrintToPDF

Usage

```
PrintToPDF(
  Path,
  OutputName,
  ObjectList = NULL,
  Tables = FALSE,
  MaxPages = 500,
  Title = "Model Output",
  Width = 12,
  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 12
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

ReactiveLoadCSV   *ReactiveLoadCSV*

Description

Use this function to import csv's, track the time it was imported, and remove other objects

Usage

```
ReactiveLoadCSV(
  input,
  InputVal = NULL,
  ProjectList = NULL,
  DateUpdateName = NULL,
  RemoveObjects = NULL
)
```

Arguments

input	Passthrough
InputVal	Values that follows input$
ProjectList	Supply the project list if available. NULL otherwise
DateUpdateName	Supply the name for the ProjectList to store the import time
RemoveObjects	List of objects to remove

Author(s)

Adrian Antico

See Also

Other Shiny: `ArgNullCheck2()`, `ArgNullCheck()`, `BlankRow()`, `DateInput()`, `GenerateEvaluationMetrics()`, `NumericInput()`, `PickerInput_GetLevels()`, `PickerInput()`, `PreparePlotData()`, `ReturnParam()`, `StoreArgs()`, `TextInput()`, `observeEventLoad()`

RedYellowGreen *RedYellowGreen*

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: AutoShapeShap(), CumGainsChart(), EvalPlot(), ParDepCalPlots(), ROCPlot(), ResidualPlots(), SingleRowShapeShap(), 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)
```

ResidualOutliers   *ResidualOutliers*

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 = NULL,
  TargetColName = NULL,
  PredictedColName = NULL,
  TimeUnit = "day",
  Lags = 5,
  Diff = 1,
  MA = 5,
  SLags = 0,
  SDiff = 1,
  SMA = 0,
  tstat = 2,
  FixedParams = FALSE
)
```

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
Diff	The largest d value for differencing
MA	Max moving average
SLags	Max seasonal lags
SDiff	The largest d value for seasonal differencing
SMA	Max seasonal moving averages
tstat	the t-stat value for tsoutliers
FixedParams	Set to TRUE or FALSE. If TRUE, a stats::Arima() model if fitted with those parameter values. If FALSE, then an auto.arima is built with the parameter values representing the max those values can be.

Value

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

Author(s)

Adrian Antico

See Also

Other Unsupervised Learning: AutoClusteringScoring(), AutoClustering(), GenTSAnomVars(), H2OIsolationForestScoring(), 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.table::setorderv(x = data, cols = 'DateTime', 1)
data[, Predicted := as.numeric(
  stats::filter(
    rnorm(1000, mean = 50, sd = 20),
    filter=rep(1,10),
    circular=TRUE))]
Output <- ResidualOutliers(
  data = data,
  DateColName = "DateTime",
  TargetColName = "Target",
  PredictedColName = NULL,
  TimeUnit = "day",
  Lags = 5,
  Diff = 1,
  MA = 5,
  SLags = 0,
```

```
  SDiff = 0,
  SMA = 0,
  tstat = 4)
data <- Output[['FullData']]
model <- Output[['ARIMA_MODEL']]
outliers <- data[type != "<NA>"]

## End(Not run)
```

ResidualPlots   *ResidualPlots*

Description

Residual plots for regression models

Usage

```
ResidualPlots(
  TestData = NULL,
  Target = "Adrian",
  Predicted = "Independent_Variable1",
  DateColumnName = NULL,
  Gam_Fit = FALSE
)
```

Arguments

TestData	= NULL,
Target	= "Adrian",
Predicted	= "Independent_Variable1",
DateColumnName	"DateTime"
Gam_Fit	= TRUE

Author(s)

Adrian Antico

See Also

Other Model Evaluation and Interpretation: AutoShapeShap(), CumGainsChart(), EvalPlot(), ParDepCalPlots(), ROCPlot(), RedYellowGreen(), SingleRowShapeShap(), threshOptim()

Examples

```
## Not run:
# Create fake data
test_data <- RemixAutoML::FakeDataGenerator(
  Correlation = 0.80,
  N = 250000,
  ID = 0,
  FactorCount = 0,
```

```
  AddDate = TRUE,
  AddComment = FALSE,
  AddWeightsColumn = FALSE,
  ZIP = 0)

# Build Plots
output <- RemixAutoML::ResidualPlots(
  TestData = test_data,
  Target = "Adrian",
  Predicted = "Independent_Variable1",
  DateColumnName = "DateTime",
  Gam_Fit = TRUE)

## End(Not run)
```

ReturnParam   *Save VarName values within a project*

Description

Automatically save VarNames to project list

Usage

```
ReturnParam(input, VarName, Type, Default, Switch = FALSE)
```

Arguments

input	This is the input value within a Shiny context
VarName	The name of the VarNameument you want to store
Type	"character" "numeric" "logical"
Default	default value
Switch	= FALSE

Value

Updates ProjectList inside function

Author(s)

Adrian Antico

See Also

Other Shiny: `ArgNullCheck2()`, `ArgNullCheck()`, `BlankRow()`, `DateInput()`, `GenerateEvaluationMetrics()`, `NumericInput()`, `PickerInput_GetLevels()`, `PickerInput()`, `PreparePlotData()`, `ReactiveLoadCSV()`, `StoreArgs()`, `TextInput()`, `observeEventLoad()`

Examples

```
## Not run:
Aggregate <- RemixAutoML::ReturnParam(input, VarName = "TS_AggregateFunction", Type = "character", Default = "

## End(Not run)
```

ROCPlot *ROCPlot*

Description

Internal usage for classification methods. Returns an ROC plot

Usage

```
ROCPlot(
  data = ValidationData,
  TargetName = TargetColumnName,
  SavePlot = SaveModelObjects,
  Name = ModelID,
  metapath = metadata_path,
  modelpath = model_path
)
```

Arguments

data	validation data
TargetName	Target variable name
SavePlot	TRUE or FALSE
Name	Name for saving
metapath	Passthrough
modelpath	Passthrough

Value

ROC Plot for classification models

Author(s)

Adrian Antico

See Also

Other Model Evaluation and Interpretation: [AutoShapeShap](), [CumGainsChart](), [EvalPlot](),
[ParDepCalPlots](), [RedYellowGreen](), [ResidualPlots](), [SingleRowShapeShap](), [threshOptim]()

RunRemixAutoML   *RunRemixAutoML*

Description

GUI for running RemixAutoML functions. Time series evaluation and forecasting, machine learning, etc.

Usage

```
RunRemixAutoML(
  data = NULL,
  ModelOutput = NULL,
  TargetName = NULL,
  PredictName = NULL,
  DateName = NULL,
  Debug = FALSE
)
```

Author(s)

Adrian Antico

See Also

Other GUI: `AppsPlotting()`

ScatterCopula   *ScatterCopula*

Description

Dual plot. One on original scale and one using empirical copula data

Usage

```
ScatterCopula(
  data = NULL,
  x_var = NULL,
  y_var = NULL,
  GroupVariable = NULL,
  FacetCol = NULL,
  FacetRow = NULL,
  SizeVar1 = NULL,
  SampleCount = 100000L,
  FitGam = TRUE,
  color = "darkblue",
  point_size = 0.5,
  text_size = 12,
  x_axis_text_angle = 35,
```

```
    y_axis_text_angle = 0,
    chart_color = "lightsteelblue1",
    border_color = "darkblue",
    text_color = "darkblue",
    grid_color = "white",
    background_color = "gray95",
    legend_position = "bottom"
)
```

Arguments

`data`	Source data.table
`x_var`	Numeric variable
`y_var`	Numeric variable
`GroupVariable`	Color options
`FacetCol`	NULL or string
`FacetRow`	NULL or string
`SizeVar1`	NULL. Use to size the dots by a variable
`SampleCount`	Number of randomized rows to utilize. For speedup and memory purposes
`FitGam`	Add gam fit to scatterplot and copula plot
`color`	= "darkblue"
`point_size`	= 0.50
`text_size`	= 12
`x_axis_text_angle`	= 35
`y_axis_text_angle`	= 0
`chart_color`	= "lightsteelblue1"
`border_color`	= "darkblue"
`text_color`	= "darkblue"
`grid_color`	= "white"
`background_color`	= "gray95"
`legend_position`	= "bottom

Author(s)

Adrian Antico

See Also

Other EDA: `AutoWordFreq()`, `EDA_Histograms()`, `Mode()`, `PlotGUI()`, `UserBaseEvolution()`

Examples

```
## Not run:
# Create data
data <- RemixAutoML::FakeDataGenerator()

# Build plot
RemixAutoML::ScatterCopula(
  data = data,
  x_var = 'Independent_Variable1',
  y_var = 'Independent_Variable2',
  GroupVariable = NULL, #'Factor_1',
  FacetCol = 'Factor_1',
  FacetRow = NULL,
  SizeVar1 = 'Independent_Variable1',
  SampleCount = 100000L,
  FitGam = FALSE,
  color = "darkblue",
  point_size = 0.50,
  text_size = 12,
  x_axis_text_angle = 35,
  y_axis_text_angle = 0,
  chart_color = "lightsteelblue1",
  border_color = "darkblue",
  text_color = "darkblue",
  grid_color = "white",
  background_color = "gray95",
  legend_position = "bottom")

## End(Not run)
```

SingleRowShapeShap   *SingleRowShapeShap*

Description

SingleRowShapeShap will convert a single row of your shap data into a table

Usage

```
SingleRowShapeShap(ShapData = NULL, EntityID = NULL, DateColumnName = NULL)
```

Arguments

ShapData   Scoring data from AutoCatBoostScoring with classification or regression

Author(s)

Adrian Antico

See Also

Other Model Evaluation and Interpretation: AutoShapeShap(), CumGainsChart(), EvalPlot(),
ParDepCalPlots(), ROCPlot(), RedYellowGreen(), ResidualPlots(), threshOptim()

SQL_ClearTable *SQL_ClearTable*

Description

SQL_ClearTable remove all rows from a database table

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 Database: [AutoDataDictionaries()](), [SQL_DropTable()](), [SQL_Query_Push()](), [SQL_Query()](),
[SQL_SaveTable()](), [SQL_Server_DBConnection()]()

SQL_DropTable *SQL_DropTable*

Description

SQL_DropTable drop a database table

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 Database: [AutoDataDictionaries()](), [SQL_ClearTable()](), [SQL_Query_Push()](), [SQL_Query()](),
[SQL_SaveTable()](), [SQL_Server_DBConnection()]()

SQL_Query *SQL_Query*

Description

SQL_Query get data from a database table

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 Database: [AutoDataDictionaries()](), [SQL_ClearTable()](), [SQL_DropTable()](), [SQL_Query_Push()](),
[SQL_SaveTable()](), [SQL_Server_DBConnection()]()

SQL_Query_Push *SQL_Query_Push*

Description

SQL_Query_Push push data to a database table

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 Database: `AutoDataDictionaries()`, `SQL_ClearTable()`, `SQL_DropTable()`, `SQL_Query()`, `SQL_SaveTable()`, `SQL_Server_DBConnection()`

SQL_SaveTable *SQL_SaveTable*

Description

SQL_SaveTable create a database table

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 Database: `AutoDataDictionaries()`, `SQL_ClearTable()`, `SQL_DropTable()`, `SQL_Query_Push()`, `SQL_Query()`, `SQL_Server_DBConnection()`

SQL_Server_DBConnection

SQL_Server_DBConnection

Description

SQL_Server_DBConnection makes a connection to a sql server database

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 Database: `AutoDataDictionaries()`, `SQL_ClearTable()`, `SQL_DropTable()`, `SQL_Query_Push()`, `SQL_Query()`, `SQL_SaveTable()`

StoreArgs   *Store Args values within a project*

Description

Automatically save arguments to project list

Usage

```
StoreArgs(input, ProjectList, VarName, Type, Default)
```

Arguments

`input`	This is the input value within a Shiny context
`ProjectList`	This is the VarNameument collection list
`VarName`	The name of the VarNameument you want to store
`Type`	"character" "numeric" "logical"
`Default`	default value that gets saved

Value

Updates ProjectList inside function. Do not assign function to anything

Author(s)

Adrian Antico

See Also

Other Shiny: `ArgNullCheck2()`, `ArgNullCheck()`, `BlankRow()`, `DateInput()`, `GenerateEvaluationMetrics()`, `NumericInput()`, `PickerInput_GetLevels()`, `PickerInput()`, `PreparePlotData()`, `ReactiveLoadCSV()`, `ReturnParam()`, `TextInput()`, `observeEventLoad()`

Examples

```
## Not run:
StoreVarNames(input, ProjectList, "NTrees", "numeric", 1000)

## End(Not run)
```

TextInput   *TextInput*

Description

TextInput automatically builds a text input with ProjectList argument usage if it exists

Usage

```
TextInput(
  InputID = "TS_CARMA_HolidayMovingAverages",
  Label = "Path to Data",
  Value = NULL,
  Width = "100%",
  Placeholder = "NULL"
)
```

Arguments

InputID	Feeds ProjectList and inputId. Argument saved in ProjectList
Label	Feeds label
Value	Default
Width	width arg
Format	Date format

Value

PickerInput object for server.R to go into renderUI(PickerInput())

Author(s)

Adrian Antico

See Also

Other Shiny: [ArgNullCheck2()](#), [ArgNullCheck()](#), [BlankRow()](#), [DateInput()](#), [GenerateEvaluationMetrics()](#), [NumericInput()](#), [PickerInput_GetLevels()](#), [PickerInput()](#), [PreparePlotData()](#), [ReactiveLoadCSV()](#), [ReturnParam()](#), [StoreArgs()](#), [observeEventLoad()](#)

Examples

```
## Not run:
output$TS_Date <- renderUI({
  RemixAutoML::TextInput(InputID = "TS_CARMA_HolidayMovingAverages",
                         Label = "Import Data Creation Date",
                         Value = Sys.Date(),
                         Placeholder = "yyyy-mm-dd")})

## End(Not run)
```

threshOptim	*threshOptim*

Description

threshOptim 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: AutoShapeShap(), CumGainsChart(), EvalPlot(), ParDepCalPlots(), ROCPlot(), RedYellowGreen(), ResidualPlots(), SingleRowShapeShap()

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*

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

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*

Description

TimeSeriesFill For Completing Time Series Data For Single Series or Time Series by Group

Usage

```
TimeSeriesFill(
  data = data,
  DateColumnName = ”Date”,
  GroupVariables = c(”Store”, ”Dept”),
  TimeUnit = ”weeks”,
  FillType = c(”maxmax”, ”minmax”, ”maxmin”, ”minmin”),
  MaxMissingPercent = 0.05,
  SimpleImpute = FALSE
)
```

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 maxmax - Fill from the absolute min date to the absolute max date, minmax - Fill from the max date of the min set to the absolute max date, maxmin - Fill from the absolute min date to the min of the max dates, or minmin - Fill from the max date of the min dates to the min date of the max dates
MaxMissingPercent	
	The maximum amount of missing values an individual series can have to remain and be imputed. Otherwise, they are discarded.
SimpleImpute	Set to TRUE or FALSE. With TRUE numeric cols will fill NAs with a -1 and non-numeric cols with a "0"

Value

Returns a data table with missing time series records filled (currently just zeros)

Author(s)

Adrian Antico

See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoHierarchicalFourier(),
AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(),
AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring(),
CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(),
H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep()

Examples

```
## Not run:

# Pull in data
data <- data <- data.table::fread("https://www.dropbox.com/s/2str3ek4f4cheqi/walmart_train.csv?dl=1")

# Run function
data <- TimeSeriesFill(
  data,
  DateColumnName = "Date",
  GroupVariables = c("Store","Dept"),
  TimeUnit = "weeks",
  FillType = "maxmax",
  SimpleImpute = FALSE)

## End(Not run)
```

UniqueLevels *UniqueLevels*

Description

UniqueLevels

Usage

```
UniqueLevels(input, data, n, GroupVars = NULL)
```

Arguments

input	passthrough
data	data.table
GroupVars	passthrough

UserBaseEvolution *UserBaseEvolution*

Description

This function creates a table of user counts over time for accumulated unique users, active unique users, new unique users, retained unique users, churned unique users, and reactivated unique users. You can run this with several specifications. You can request monthly, weekly, or daily counts and you can specify a churn window for the computations. If you want to compare how many churned users also churned from another segment of sorts, provide a list in the Cross parameter.

Usage

```
UserBaseEvolution(
  data,
  Cross = NULL,
  Entity = NULL,
  DateColumnName = NULL,
  TimeAgg = NULL,
  ChurnPeriods = 1
)
```

Arguments

data	Source data.table
Cross	Can be NULL. User base from non source. Must be a named list. Names of list are used to name columns in output table. Entity and DateColumnName must be identical across data sets.
Entity	Column name of the entity / user
DateColumnName	Name of the date column used for inclusion of users in time periods
TimeAgg	Choose from 'Month', 'Week', or 'Day'. Do not lowercase
ChurnPeriods	Defaults to 1. This means for TimeAgg = 'Month' a one month churn period is used. For TimeAgg = 'Week' you will have a one week churn period. If you set ChurnPeriods to 2 then it will be a 2 month churn or a 2 week churn. Same logic applies for daily.

Author(s)

Adrian Antico

See Also

Other EDA: AutoWordFreq(), EDA_Histograms(), Mode(), PlotGUI(), ScatterCopula()

---

withConsoleRedirect *withConsoleRedirect*

Description

withConsoleRedirect

Usage

```
withConsoleRedirect(containerId, expr)
```

Arguments

containerId	Passthrough
expr	Code

Examples

```
## Not run:
library(shiny)
Example usage

ui <- fluidPage(
  pre(id = "console")
)

server <- function(input, output, session) {
  observe({
    invalidateLater(1)

    withConsoleRedirect("console", {
      str(cars)
    })
  })
}

shinyApp(ui, server)

## End(Not run)
```

Index