Package ‘AutoQuant’

March 26, 2023

Title AutoQuant

Version 1.0.0

Date 2022-12-1

Maintainer Adrian Antico <adrianantico@gmail.com>

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

License MPL-2.0 | file LICENSE

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

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

Depends R (>= 3.5.0)

Imports bit64, data.table, doParallel, foreach, lubridate, timeDate, Rodeo

Suggests knitr, rmarkdown, gridExtra

VignetteBuilder knitr

Contact Adrian Antico

Encoding UTF-8

Language en-US

LazyData true

NeedsCompilation no

RoxygenNote 7.2.1

SystemRequirements Java (>= 7.0)

Author Adrian Antico [aut, cre]

ByteCompile TRUE

R topics documented:

`RemixAutoML-package` *Automated Machine Learning Remixed*

Description

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

Details

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

Author(s)

Adrian Antico, adrianantico@gmail.com, Douglas Pestana

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,
  EncodeMethod = "credibility",
  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(0, 1, 1, 0),
  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', '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.

`EncodeMethod`
: 'credibility', 'binary', 'm_estimator', 'woe', 'target_encoding', 'poly_encode', 'backward_difference', 'helmert'

`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 <- AutoQuant::FakeDataGenerator(
  Correlation = 0.85,
  N = 10000,
  ID = 2,
  ZIP = 0,
  AddDate = FALSE,
  Classification = TRUE,
  MultiClass = FALSE)

# Run function
TestModel <- AutoQuant::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'),
```

```
    EncodeMethod = 'credibility',

    # Evaluation args
    ClassWeights = c(1L,1L),
    CostMatrixWeights = c(0,1,1,0),
    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)
```

`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,
  EncodeMethod = "credibility",
  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', '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.

`EncodeMethod`
: 'binary', 'm_estimator', 'credibility', 'woe', 'target_encoding', 'poly_encode', 'backward_difference', 'helmert'

`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 <- AutoQuant::FakeDataGenerator(
  Correlation = 0.85,
  N = 10000L,
  ID = 2L,
  ZIP = 0L,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = TRUE)

# Run function
TestModel <- AutoQuant::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'),
EncodeMethod = 'credibility',

# 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"),
  ReturnShap = TRUE,
  data = NULL,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  PrimaryDateColumn = NULL,
  WeightsColumnName = NULL,
  IDcols = NULL,
  EncodeMethod = "credibility",
  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', 'Score_TrainData')

ReturnShap
: TRUE. Set to FALSE to not generate shap values.

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.

EncodeMethod
: '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 '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, Evalution-Plot.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()`, `AutoXGBoostRegression()`

Examples

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

# Run function
TestModel <- AutoQuant::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'),
    EncodeMethod = 'credibility',
    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,
  Debug = 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 CharacterEncode()
`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

Debug
: = FALSE

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 <- AutoQuant::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 <- AutoQuant::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 <- AutoQuant::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)

  # Step through scoring function
  library(AutoQuant)
  library(data.table)
  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
  Debug = 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: PostGRE_AppendData(), PostGRE_CreateTable(), PostGRE_GetTableNames(), PostGRE_ListTables(), PostGRE_Query(), PostGRE_RemoveCreateAppend(), PostGRE_RemoveTable(), PosteGRE_CreateDatabase(), PosteGRE_DropDB(), PosteGRE_ListDatabases(), SQL_ClearTable(), SQL_DropTable(), SQL_Query_Push(), SQL_Query(), SQL_SaveTable(), SQL_Server_DBConnection()

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", "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 "EvalMetrics", "Score_TrainData", "h2o.explain"
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 Positive 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, EvalutionPlot.png, EvaluationMetrics.csv, ParDepPlots.R a named list of features with partial dependence calibration plots, GridCollect, and GridList

Author(s)

Adrian Antico

See Also

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

Examples

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

TestModel <- AutoQuant::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","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)
```

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", "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 "EvalMetrics", "Score_TrainData", "h2o.explain"
- `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, EvaluationMetrics.csv, GridCollect, and GridList

Author(s)

Adrian Antico

See Also

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

Examples

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

# Run function
TestModel <- AutoQuant::AutoH2oDRFMultiClass(
  OutputSelection = c("EvalMetrics","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", inte
  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", "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),
  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("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 "EvalMetrics", "Score_TrainData", "h2o.explain"

`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". Other options include "Enum", "OneHotInternal", "OneHotExplicit", "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimite"

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(), AutoXGBoostRegression()

Examples

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

# Run function
TestModel <- AutoQuant::AutoH2oDRFRegression(

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

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

  # Metadata arguments
  OutputSelection = c("EvalMetrics","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("Asinh","Asin","Log","LogPlus1", "Sqrt","Logit"),

  # 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", "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", "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 <- AutoQuant::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 <- AutoQuant::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", "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", "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", "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, EvaluationMetrics.csv, GridCollect, and GridList

Author(s)

Adrian Antico

See Also

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

Examples

```
# Create some dummy correlated data with numeric and categorical features
data <- AutoQuant::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 <- AutoQuant::AutoH2oGAMMultiClass(
    OutputSelection = c("EvalMetrics", "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", inte
    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", "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", "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, EvalutionPlot.png, EvalutionBoxPlot.png, EvaluationMetrics.csv, ParDepPlots.R a named list of features with partial dependence calibration plots, ParDepBoxPlots.R, GridCollect, GridList, and Transformation metadata

Author(s)

Adrian Antico

See Also

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

Examples

```
# Create some dummy correlated data
data <- AutoQuant::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 <- AutoQuant::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", "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", "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", "Score_TrainData", "h2o.explain")

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, EvalutionPlot.png, EvaluationMetrics.csv, ParDepPlots.R a named list of features with partial dependence calibration plots, GridCollect, and GridList

Author(s)

Adrian Antico

See Also

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

Examples

```
## Not run:
# Create some dummy correlated data
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.85,
```

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

TestModel <- AutoQuant::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", "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)
```

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", "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", "Score_TrainData", "h2o.explain")
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 <- AutoQuant::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000,
  ID = 2,
  ZIP = 0,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = TRUE)

# Run function
TestModel <- AutoQuant::AutoH2oGBMMultiClass(
  OutputSelection = c("EvalMetrics", "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", inte
  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", "Score_TrainData"),
  data = NULL,
  TrainOnFull = FALSE,
  ValidationData = NULL,
  TestData = NULL,
  TargetColumnName = NULL,
  FeatureColNames = NULL,
  WeightsColumn = NULL,
  TransformNumericColumns = NULL,
  Methods = c("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", "Score_TrainData", "h2o.explain")

`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, EvalutionPlot.png, EvalutionBoxPlot.png, EvaluationMetrics.csv, ParDepPlots.R a named list of features with partial dependence calibration plots, ParDepBoxPlots.R, GridCollect, GridList, and metadata

Author(s)

Adrian Antico

See Also

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

Examples

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

# Run function
TestModel <- AutoQuant::AutoH2oGBMRegression(

  # Compute management
 MaxMem = {gc();paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo", inte
  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("Asinh", "Asin", "Log", "LogPlus1", "Sqrt", "Logit"),

  # 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", "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", "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. You can also pass character names of the columns.

`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"

`Link` identity, logit, log, inverse, tweedie

`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

Value

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

Author(s)

Adrian Antico

See Also

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

Examples

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

# Run function
TestModel <- AutoQuant::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", "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", "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", "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. You can also pass character names of the columns.
`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"

Link — "family_default"

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

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 <- AutoQuant::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000L,
  ID = 2L,
  ZIP = 0L,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = TRUE)

# Run function
TestModel <- AutoQuant::AutoH2oGLMMultiClass(

  # Compute management
  OutputSelection = c("EvalMetrics", "Score_TrainData"),
 MaxMem = {gc();paste0(as.character(floor(as.numeric(system("awk '/MemFree/ {print $2}' /proc/meminfo", inte
  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", "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 "EvalMetrics", "Score_TrainData", "h2o.explain"

`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. You can also pass character names of the columns.

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
: 1, 2, 3 for Poisson, Gamma, and Gaussian

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, EvalutionPlot.png, EvalutionBoxPlot.png, EvaluationMetrics.csv, ParDepPlots.R a named list of features with partial dependence calibration plots, ParDepBoxPlots.R, GridCollect, GridList, and Transformation metadata

Author(s)

Adrian Antico

See Also

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

Examples

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

# Run function
TestModel <- AutoQuant::AutoH2oGLMRegression(

  # 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", "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("Asinh", "Asin", "Log", "LogPlus1", "Sqrt", "Logit"),

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

Usage

```
AutoH2oMLClassifier(
  OutputSelection = c("EvalMetrics", "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", "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)

`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, EvalutionPlot.png, EvaluationMetrics.csv, ParDepPlots.R a named list of features with partial dependence calibration plots, GridCollect, and GridList

Author(s)

Adrian Antico

See Also

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

Examples

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

TestModel <- AutoQuant::AutoH2oMLClassifier(
  OutputSelection = c("EvalMetrics", "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", inte
  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", "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", "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, EvaluationMetrics.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 <- AutoQuant::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000,
  ID = 2,
  ZIP = 0,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = TRUE)

# Run function
TestModel <- AutoQuant::AutoH2oMLMultiClass(
  OutputSelection = c("EvalMetrics", "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", inte
  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", "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", "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(), AutoXGBoostRegression()

Examples

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

# Run function
TestModel <- AutoQuant::AutoH2oMLRegression(

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

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

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

`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

CostMatrixWeights
: = c(1,0,0,1)

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", "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, 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 - Binary Classification: AutoCatBoostClassifier(), AutoH2oDRFClassifier(), AutoH2oGAMClassifier(), AutoH2oGBMClassifier(), AutoH2oGLMClassifier(), AutoH2oMLClassifier(), AutoXGBoostClassifier()

Examples

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

# Run function
TestModel <- AutoQuant::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,
  CostMatrixWeights = c(1,0,0,1),
  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)
```

`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", "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, 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

Examples

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

# Run function
TestModel <- AutoQuant::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', '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' (or 'mean_squared_error'), 'regression_l1' (or 'mean_absolute_error'), 'mae' (or 'mean_absolute_percentage_error'), 'huber', 'fair', 'poisson', 'quantile', 'gamma', 'tweedie'

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, 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()`, `AutoXGBoostRegression()`

Examples

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

# Run function
TestModel <- AutoQuant::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 <- AutoQuant::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)
```

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()

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(), 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)
```

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(0, 1, 1, 0),
  grid_eval_metric = "MCC",
  eval_metric = "auc",
  TreeMethod = "hist",
  GridTune = FALSE,
```

```
  BaselineComparison = "default",
  MaxModelsInGrid = 10L,
  MaxRunsWithoutNewWinner = 20L,
  MaxRunMinutes = 24L * 60L,
  PassInGrid = NULL,
  early_stopping_rounds = 100L,
  Trees = 1000L,
  num_parallel_tree = 1,
  eta = 0.3,
  max_depth = 9,
  min_child_weight = 1,
  subsample = 1,
  colsample_bytree = 1,
  DebugMode = FALSE,
  alpha = 0,
  lambda = 1
)
```

Arguments

OutputSelection	You can select what type of output you want returned. Choose from c("Importances", "EvalPlots", "EvalMetrics", "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.

early_stopping_rounds
: = 100L

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)

num_parallel_tree
: = 1. If setting greater than 1, set colsample_bytree < 1, subsample < 1 and round = 1

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

alpha
: 0. L1 Reg.

lambda
: 1. L2 Reg.

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 <- AutoQuant::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000L,
  ID = 2L,
  ZIP = 0L,
  AddDate = FALSE,
  Classification = TRUE,
  MultiClass = FALSE)

# Run function
TestModel <- AutoQuant::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(0,1,1,0),
  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)
```

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,
  early_stopping_rounds = 100L,
  Trees = 50L,
  num_parallel_tree = 1,
  eta = NULL,
  max_depth = NULL,
  min_child_weight = NULL,
  subsample = NULL,
  colsample_bytree = NULL,
  alpha = 0,
  lambda = 1
)
```

Arguments

`OutputSelection`
: You can select what type of output you want returned. Choose from c("Importances", "EvalPlots", "EvalMetrics", "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 'merror' or 'mlogloss'

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.

`early_stopping_rounds`
: = 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)

`num_parallel_tree`
: = 1. If setting greater than 1, set colsample_bytree < 1, subsample < 1 and round = 1

`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)

`alpha`
: 0. L1 Reg.

`lambda`
: 1. L2 Reg.

Value

Saves to file and returned in list: VariableImportance.csv, Model, ValidationData.csv, EvaluationMetrics.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 <- AutoQuant::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000L,
  ID = 2L,
  ZIP = 0L,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = TRUE)

# Run function
TestModel <- AutoQuant::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,
  early_stopping_rounds = 100L,
  Trees = 50L,
  num_parallel_tree = 1,
  eta = NULL,
  max_depth = NULL,
  min_child_weight = NULL,
  subsample = NULL,
  colsample_bytree = NULL,
  alpha = 0,
  lambda = 1
)
```

Arguments

OutputSelection	You can select what type of output you want returned. Choose from c("Importances", "EvalPlots", "EvalMetrics", "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 "rmse", "mae", "mape"

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.

early_stopping_rounds
= 100L

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)

num_parallel_tree
: = 1. If setting greater than 1, set colsample_bytree < 1, subsample < 1 and round = 1

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)

alpha
: 0. L1 Reg.

lambda
: 1. L2 Reg.

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()

Examples

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

# Run function
TestModel <- AutoQuant::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 ModelDataPrep() 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)
```

`BarPlot` *BarPlot*

Description

Build a bar plot by simply passing arguments to a single function. It will sample your data using SampleSize number of rows. Sampled data is randomized.

Usage

```
BarPlot(
  data = NULL,
  XVar = NULL,
  YVar = NULL,
  AggMethod = "mean",
  ColorVar = NULL,
  FacetVar1 = NULL,
  FacetVar2 = NULL,
  SampleSize = 1000000L,
  FillColor = "gray",
```

```
  YTicks = "Default",
  XTicks = "Default",
  TextSize = 12,
  AngleX = 90,
  AngleY = 0,
  ChartColor = "lightsteelblue1",
  BorderColor = "darkblue",
  TextColor = "darkblue",
  GridColor = "white",
  BackGroundColor = "gray95",
  SubTitleColor = "blue",
  LegendPosition = "bottom",
  LegendBorderSize = 0.5,
  LegendLineType = "solid",
  Debug = FALSE
)
```

Arguments

data	Source data.table
XVar	Column name of X-Axis variable. If NULL then ignored
YVar	Column name of Y-Axis variable. If NULL then ignored
AggMethod	Choose from 'mean', 'sum', 'sd', and 'median'
ColorVar	Column name of Group Variable for distinct colored histograms by group levels
FacetVar1	Column name of facet variable 1. If NULL then ignored
FacetVar2	Column name of facet variable 2. If NULL then ignored
SampleSize	An integer for the number of rows to use. Sampled data is randomized. If NULL then ignored
FillColor	'gray'
YTicks	Choose from 'Default', 'Percentiles', 'Every 5th percentile', 'Deciles', 'Quantiles', 'Quartiles'
XTicks	Choose from 'Default', '1 year', '1 day', '3 day', '1 week', '2 week', '1 month', '3 month', '6 month', '2 year', '5 year', '10 year', '1 minute', '15 minutes', '30 minutes', '1 hour', '3 hour', '6 hour', '12 hour'
TextSize	14
AngleX	90
AngleY	0
ChartColor	'lightsteelblue'
BorderColor	'darkblue'
TextColor	'darkblue'
GridColor	'white'
BackGroundColor	'gray95'
SubTitleColor	'darkblue'
LegendPosition	'bottom'
LegendBorderSize	0.50

LegendLineType	'solid'
Debug	FALSE
OutlierSize	0.10
OutlierColor	'blue'

Author(s)

Adrian Antico

See Also

Other Graphics: AddFacet(), BoxPlot(), ChartTheme(), CorrMatrixPlot(), DensityPlot(), HeatMapPlot(), HistPlot(), PlotlyConversion(), StockData(), StockPlot(), ViolinPlot(), multiplot()

Examples

```
## Not run:
# Load packages
library(AutoQuant)
library(data.table)

# Load data
data <- data.table::fread(file = file.path('C:/Users/Bizon/Documents/GitHub/BenchmarkData1.csv'))

# Run function
AutoQuant:::BarPlot(
  data = data,
  XVar = 'Region',
  YVar = 'Weekly_Sales',
  AggMethod = 'mean',
  ColorVar = NULL,
  FacetVar1 = 'Store',
  FacetVar2 = 'Dept',
  SampleSize = 1000000L,
  FillColor = 'gray',
  YTicks = 'Default',
  XTicks = 'Default',
  TextSize = 12,
  AngleX = 90,
  AngleY = 0,
  ChartColor = 'lightsteelblue1',
  BorderColor = 'darkblue',
  TextColor = 'darkblue',
  GridColor = 'white',
  BackGroundColor = 'gray95',
  SubTitleColor = 'blue',
  LegendPosition = 'bottom',
  LegendBorderSize = 0.50,
  LegendLineType = 'solid',
  Debug = FALSE)

# Step through function
# XVar = 'Region'
# YVar = 'Weekly_Sales'
```

```
# AggMethod = 'mean'
# ColorVar = NULL
# FacetVar1 = NULL
# FacetVar2 = NULL
# SampleSize = 1000000L
# FillColor = 'gray'
# YTicks = 'Default'
# XTicks = 'Default'
# TextSize = 12
# AngleX = 90
# AngleY = 0
# ChartColor = 'lightsteelblue1'
# BorderColor = 'darkblue'
# TextColor = 'darkblue'
# GridColor = 'white'
# BackGroundColor = 'gray95'
# SubTitleColor = 'blue'
# LegendPosition = 'bottom'
# LegendBorderSize = 0.50
# LegendLineType = 'solid'
# Debug = FALSE

## End(Not run)
```

BenchmarkData *BenchmarkData*

Description

Modified version of h2o datatable benchmark data

Usage

```
BenchmarkData(
  NRows = 1e+07,
  Levels = 1e+06,
  NAs = -1L,
  FixedEffects = c(5, 10, 15),
  CharVars = TRUE,
  IntVars = TRUE,
  Sort = TRUE
)
```

Arguments

NAs	= -1L
FixedEffects	c(5,10,15), number of levels for each variable
CharVars	FALSE
IntVars	TRUE
Sort	= TRUE
N	= 10000000,

RandomLevels	= 1000000
RandomEffects	c(3)

BoxPlot *BoxPlot*

Description

Build a box plot by simply passing arguments to a single function. It will sample your data using SampleSize number of rows. Sampled data is randomized.

Usage

```
BoxPlot(
  data = NULL,
  XVar = NULL,
  YVar = NULL,
  FacetVar1 = NULL,
  FacetVar2 = NULL,
  SampleSize = 1000000L,
  FillColor = "gray",
  OutlierSize = 0.1,
  OutlierColor = "blue",
  YTicks = "Default",
  XTicks = "Default",
  TextSize = 12,
  AngleX = 90,
  AngleY = 0,
  ChartColor = "lightsteelblue1",
  BorderColor = "darkblue",
  TextColor = "darkblue",
  GridColor = "white",
  BackGroundColor = "gray95",
  SubTitleColor = "blue",
  LegendPosition = "bottom",
  LegendBorderSize = 0.5,
  LegendLineType = "solid",
  Debug = FALSE
)
```

Arguments

data	Source data.table
XVar	Column name of X-Axis variable. If NULL then ignored
YVar	Column name of Y-Axis variable. If NULL then ignored
FacetVar1	Column name of facet variable 1. If NULL then ignored
FacetVar2	Column name of facet variable 2. If NULL then ignored
SampleSize	An integer for the number of rows to use. Sampled data is randomized. If NULL then ignored

`FillColor`	'gray'
`OutlierSize`	0.10
`OutlierColor`	'blue'
`YTicks`	Choose from 'Default', 'Percentiles', 'Every 5th percentile', 'Deciles', 'Quantiles', 'Quartiles'
`XTicks`	Choose from 'Default', '1 year', '1 day', '3 day', '1 week', '2 week', '1 month', '3 month', '6 month', '2 year', '5 year', '10 year', '1 minute', '15 minutes', '30 minutes', '1 hour', '3 hour', '6 hour', '12 hour'
`TextSize`	14
`AngleX`	90
`AngleY`	0
`ChartColor`	'lightsteelblue'
`BorderColor`	'darkblue'
`TextColor`	'darkblue'
`GridColor`	'white'
`BackGroundColor`	'gray95'
`SubTitleColor`	'darkblue'
`LegendPosition`	'bottom'
`LegendBorderSize`	0.50
`LegendLineType`	'solid'
`Debug`	FALSE

Author(s)

Adrian Antico

See Also

Other Graphics: `AddFacet()`, `BarPlot()`, `ChartTheme()`, `CorrMatrixPlot()`, `DensityPlot()`, `HeatMapPlot()`, `HistPlot()`, `PlotlyConversion()`, `StockData()`, `StockPlot()`, `ViolinPlot()`, `multiplot()`

Examples

```
## Not run:
# Load packages
library(AutoQuant)
library(data.table)

# Load data
data <- data.table::fread(file = file.path('C:/Users/Bizon/Documents/GitHub/BenchmarkData1.csv'))

# Run function
AutoQuant:::BoxPlot(
  data = data,
  XVar = 'Region',
  YVar = 'Weekly_Sales',
```

```
  FacetVar1 = 'Store',
  FacetVar2 = NULL,
  SampleSize = 1000000L,
  FillColor = 'gray',
  OutlierSize = 0.10,
  OutlierColor = 'blue',
  YTicks = 'Default',
  XTicks = 'Default',
  TextSize = 12,
  AngleX = 90,
  AngleY = 0,
  ChartColor = 'lightsteelblue1',
  BorderColor = 'darkblue',
  TextColor = 'darkblue',
  GridColor = 'white',
  BackGroundColor = 'gray95',
  SubTitleColor = 'blue',
  LegendPosition = 'bottom',
  LegendBorderSize = 0.50,
  LegendLineType = 'solid',
  Debug = FALSE)

# Step through function
# XVar = 'Region'
# YVar = 'Weekly_Sales'
# FacetVar1 = 'Store'
# FacetVar2 = 'Dept'
# SampleSize = 1000000L
# FillColor = 'gray'
# OutlierSize = 0.10
# OutlierColor = 'blue'
# YTicks = 'Default'
# XTicks = 'Default'
# TextSize = 12
# AngleX = 90
# AngleY = 0
# ChartColor = 'lightsteelblue1'
# BorderColor = 'darkblue'
# TextColor = 'darkblue'
# GridColor = 'white'
# BackGroundColor = 'gray95'
# SubTitleColor = 'blue'
# LegendPosition = 'bottom'
# LegendBorderSize = 0.50
# LegendLineType = 'solid'
# Debug = FALSE

## End(Not run)
```

CausalMediation *CausalMediation*

Description

CausalMediation utilizes models from regmedint package

Usage

```
CausalMediation(
  data,
  OutcomeTargetVariable = NULL,
  TreatmentVariable = NULL,
  MediatorVariable = NULL,
  Covariates = NULL,
  MM_TreatmentCovariates = NULL,
  OM_TreatmentCovariates = NULL,
  OM_MediatorCovariates = NULL,
  SurvivalEventVariable = NULL,
  UnTreated_ReferenceIndicator = NULL,
  Treated_ReferenceIndicator = NULL,
  Mediator_ControlDirectEffectLevel = NULL,
  Covariate_NaturalDirectIndirect = 0,
  MediatorTargetType = "linear",
  OutcomeTargetType = "linear",
  TreatmentMediatorInteraction = TRUE,
  CaseControlSourceData = FALSE,
  RemoveNA = FALSE
)
```

Arguments

`data` Data frame containing the following relevant variables.

`OutcomeTargetVariable`
yvar in underlying model. A character vector of length 1. Outcome variable name. It should be the time variable for the survival outcome.

`TreatmentVariable`
avar in underlying model. A character vector of length 1. Treatment variable name.

`MediatorVariable`
mvar in underlying model. A character vector of length 1. Mediator variable name.

`Covariates` For main model

`MM_TreatmentCovariates`
emm_ac_mreg in underlying model. A character vector of length > 0. Effect modifiers names. The covariate vector in treatment-covariate product term in the mediator model.

`OM_TreatmentCovariates`
emm_ac_yreg in underlying model. A character vector of length > 0. Effect modifiers names. The covariate vector in treatment-covariate product term in the outcome model.

`OM_MediatorCovariates`
emm_mc_yreg in underlying model. A character vector of length > 0. Effect modifiers names. The covariate vector in mediator-covariate product term in outcome model.

`SurvivalEventVariable`
eventvar in underlying model. An character vector of length 1. Only required for survival outcome regression models. Note that the coding is 1 for event and 0 for censoring, following the R survival package convention.

`UnTreated_ReferenceIndicator`
: a0 in underlying model. A numeric vector of length 1. The reference level of treatment variable that is considered "untreated" or "unexposed".

`Treated_ReferenceIndicator`
: a1 in underlying model. A numeric vector of length 1.

`Mediator_ControlDirectEffectLevel`
: m_cde in underlying model. A numeric vector of length 1. Mediator level at which controlled direct effect is evaluated at.

`Covariate_NaturalDirectIndirect`
: c_cond in underlying model. A numeric vector of the same length as `cvar`. Covariate levels at which natural direct and indirect effects are evaluated at.

`MediatorTargetType`
: mreg in underlying model. A character vector of length 1. Mediator regression type: `"linear"` or `"logistic"`.

`OutcomeTargetType`
: yreg in underlying model. A character vector of length 1. Outcome regression type: `"linear"`, `"logistic"`, `"loglinear"`, `"poisson"`, `"negbin"`, `"survCox"`, `"survAFT_exp"`, or `"survAFT_weibull"`.

`TreatmentMediatorInteraction`
: interaction in underlying model. A logical vector of length 1. The presence of treatment-mediator interaction in the outcome model. Default to TRUE.

`CaseControlSourceData`
: casecontrol in underlying model. A logical vector of length 1. Default to FALSE. Whether data comes from a case-control study.

`RemoveNA`
: na_omit in underlying model. A logical vector of length 1. Default to FALSE. Whether to remove NAs in the columns of interest before fitting the models.

`ConfoundingVariables`
: cvar in underlying model. A character vector of length > 0. Covariate names. Use `NULL` if there is no covariate. However, this is a highly suspicious situation. Even if `avar` is randomized, `mvar` is not. Thus, there are usually some confounder(s) to account for the common cause structure (confounding) between `mvar` and `yvar`.

Value

list with model output object, summary output, effects output, and an effects plot

Author(s)

Adrian Antico

Examples

```
## Not run:
library(regmedint) # to load vv2015
data(vv2015)
Output <- AutoQuant::CausalMediation(
  data = vv2015,
  OutcomeTargetVariable = 'y',           # yvar char length = 0
  TreatmentVariable = "x",               # avar char length = 0 (binary)
  MediatorVariable = "m",                # mvar char length = 0 (binary)
  Covariates = "c",                      # cvar char length > 0
  MM_TreatmentCovariates = NULL,         # emm_ac_mreg = NULL char length > 0
```

```
  OM_TreatmentCovariates = NULL,          # emm_ac_yreg = NULL char length > 0
  OM_MediatorCovariates = NULL,           # emm_mc_yreg = NULL char length > 0
  SurvivalEventVariable = "event",        # eventvar char length = 0
  UnTreated_ReferenceIndicator = 0,       # ao num length = 1
  Treated_ReferenceIndicator = 1,         # a1 num length = 1
  Mediator_ControlDirectEffectLevel = 1,  # m_cde num length = 1
 Covariate_NaturalDirectIndirect = 3,   # c_cond; same length as Covariates num length = length(Covariates)
  MediatorTargetType = 'logistic',        # mreg "linear" or "logistic",
 OutcomeTargetType = 'survAFT_weibull',  # yreg "linear", "logistic", "loglinear", "poisson", "negbin", "survC
  TreatmentMediatorInteraction = TRUE,    # interaction = TRUE,
  CaseControlSourceData = FALSE,          # casecontrol = FALSE,
  RemoveNA = FALSE)

# data = vv2015
# OutcomeTargetVariable = 'y'
# TreatmentVariable = "x"
# MediatorVariable = "m"
# Covariates = "c"
# MM_TreatmentCovariates = NULL
# OM_TreatmentCovariates = NULL
# OM_MediatorCovariates = NULL
# SurvivalEventVariable = "event"
# UnTreated_ReferenceIndicator = 0
# Treated_ReferenceIndicator = 1
# Mediator_ControlDirectEffectLevel = 1
# Covariate_NaturalDirectIndirect = 3
# MediatorTargetType = 'logistic'
# OutcomeTargetType = 'survAFT_weibull'
# TreatmentMediatorInteraction = TRUE
# CaseControlSourceData = FALSE
# RemoveNA = FALSE

## End(Not run)
```

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 = 90,
  AngleY = 0,
  ChartColor = "lightsteelblue1",
  BorderColor = "darkblue",
  TextColor = "darkblue",
  SubTitleColor = "blue",
  GridColor = "white",
```

```
  BackGroundColor = "gray95",
  LegendPosition = "bottom",
  LegendBorderSize = 0.01,
  LegendLineType = "solid"
)
```

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"
SubTitleColor	'blue'
GridColor	"white"
BackGroundColor	"gray95"
LegendPosition	Where to place legend
LegendBorderSize	0.50
LegendLineType	'solid'

Value

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

Author(s)

Adrian Antico

See Also

Other Graphics: AddFacet(), BarPlot(), BoxPlot(), CorrMatrixPlot(), DensityPlot(), HeatMapPlot(), HistPlot(), PlotlyConversion(), StockData(), StockPlot(), ViolinPlot(), 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)
```

CorrMatrixPlot *CorrMatrixPlot*

Description

Build a violin plot by simply passing arguments to a single function. It will sample your data using SampleSize number of rows. Sampled data is randomized.

Usage

```
CorrMatrixPlot(data = NULL, CorrVars = NULL, Method = "spearman")
```

Arguments

data	Source data.table
CorrVars	Column names of variables you want included in the correlation matrix
Method	'spearman' default, 'pearson' otherwise

Author(s)

Adrian Antico

See Also

Other Graphics: AddFacet(), BarPlot(), BoxPlot(), ChartTheme(), DensityPlot(), HeatMapPlot(), HistPlot(), PlotlyConversion(), StockData(), StockPlot(), ViolinPlot(), multiplot()

Examples

```
## Not run:
data <- data.table::fread(file.choose())
CorrVars <- c('Weekly_Sales', 'XREG1', 'XREG2', 'XREG3')
p <- cor(data[, .SD, .SDcols = c(CorrVars)])
p1 <- heatmaply::heatmaply_cor(
  p,
  colors = c('red', 'white', 'blue'),
  xlab = "Features",
  ylab = "Features",
  k_col = 2,
  k_row = 2)

## 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()`

DataTable *DataTable*

Description

Fully loaded DT::datatable() with args prefilled

Usage

```
DataTable(data, FixedCols = 2)
```

Arguments

data	source data.table
FixedCols	Number of columns from the left to Freeze, like freeze panes in Excel. Default is 2

Author(s)

Adrian Antico

See Also

Other Shiny: DataTable2()

Examples

```
## Not run:
# Rmarkdown example of DataTable inside a <details> </Details> section

```{r Get Dependencies For DT::datatable(), echo=FALSE,include = FALSE}
# You need this code to conduct the magic dependences attaching...
DT::datatable(matrix())
```

```{js Nest All DT::datatable() inside a details drop down, echo=FALSE}
setTimeout(function() {
  var codes = document.querySelectorAll('.dataTables_wrapper');
  var code, i, d, s, p;
  for (i = 0; i < codes.length; i++) {
    code = codes[i];
    p = code.parentNode;
    d = document.createElement('details');
    s = document.createElement('summary');
    s.innerText = 'Details';
    // <details><summary>Details</summary></details>
      d.appendChild(s);
    // move the code into <details>
      p.replaceChild(d, code);
    d.appendChild(code);
  }
});
```
```

~~~
```{r Example, echo = FALSE}
AutoQuant::DataTable(data)
````

# Shiny Usage
output$Table <- shiny::renderUI({AutoQuant::DataTable(data)})


## End(Not run)
~~~

DataTable2 *DataTable2*

Description

Fully loaded DT::datatable() with args prefilled

Usage

```
DataTable2(data, FixedCols = 2L)
```

Arguments

data	source data.table
FixedCols	= 2L

Author(s)

Adrian Antico

See Also

Other Shiny: DataTable()

Examples

~~~
## Not run:
# Rmarkdown example of DataTable2 inside a <details> </Details> section

```{r Get Dependencies For DT::datatable(), echo=FALSE,include = FALSE}
# You need this code to conduct the magic dependences attaching...
DT::datatable(matrix())
```

```{js Nest All DT::datatable() inside a details drop down, echo=FALSE}
setTimeout(function() {
  var codes = document.querySelectorAll('.dataTables_wrapper');
  var code, i, d, s, p;
  for (i = 0; i < codes.length; i++) {
    code = codes[i];
    p = code.parentNode;
    d = document.createElement('details');
    s = document.createElement('summary');
~~~

```
    s.innerText = 'Details';
    // <details><summary>Details</summary></details>
      d.appendChild(s);
    // move the code into <details>
      p.replaceChild(d, code);
    d.appendChild(code);
  }
});
```
```{r Example, echo = FALSE}
AutoQuant::DataTable2(data)
````

Shiny Usage
output$Table <- shiny::renderUI({AutoQuant::DataTable2(data)})

End(Not run)
```

DensityPlot *DensityPlot*

Description

Density plots, by groups, with transparent continuous plots

Usage

```
DensityPlot(data, GroupVariables, MeasureVars)
```

Arguments

data	data.table
GroupVariables	= NULL
MeasureVariables	= NULL

See Also

Other Graphics: AddFacet(), BarPlot(), BoxPlot(), ChartTheme(), CorrMatrixPlot(), HeatMapPlot(), HistPlot(), PlotlyConversion(), StockData(), StockPlot(), ViolinPlot(), multiplot()
```

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(), 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 <- AutoQuant::FakeDataGenerator(
  Correlation = 0.70, N = 10000000, Classification = TRUE)
data.table::setnames(data, "IDcol_1", "Predict")

# Run function
AutoQuant::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:
# Create dummy data to test regression, classification, and multiclass models.
#   I don't care too much about actual relationships but I can test out on the
#   regression problem since those variables will be correlated. The binary and
#   multiclass won't however since they were created separately.

# Regression
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.77,
  N = 1000000L,
  ID = 4L,
  FactorCount = 5L,
  AddDate = TRUE,
  AddComment = TRUE,
  AddWeightsColumn = TRUE,
  ZIP = 0L,
  TimeSeries = FALSE,
  TimeSeriesTimeAgg = "day",
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Classification
data2 <- AutoQuant::FakeDataGenerator(
  Correlation = 0.77,
  N = 1000000L,
  ID = 4L,
  FactorCount = 5L,
  AddDate = TRUE,
  AddComment = TRUE,
  AddWeightsColumn = TRUE,
  ZIP = 0L,
```

```
  TimeSeries = FALSE,
  TimeSeriesTimeAgg = "day",
  ChainLadderData = FALSE,
  Classification = TRUE,
  MultiClass = FALSE)

# MultiClass
data3 <- AutoQuant::FakeDataGenerator(
  Correlation = 0.77,
  N = 1000000L,
  ID = 4L,
  FactorCount = 5L,
  AddDate = TRUE,
  AddComment = TRUE,
  AddWeightsColumn = TRUE,
  ZIP = 0L,
  TimeSeries = FALSE,
  TimeSeriesTimeAgg = "day",
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = TRUE)

data.table::setnames(data, 'Adrian', 'RegressionTarget')
data.table::setnames(data2, 'Adrian', 'BinaryTarget')
data.table::setnames(data3, 'Adrian', 'MultiClassTarget')

data <- cbind(data, data2$BinaryTarget, data3$MultiClassTarget)
data.table::setnames(data, c('V2','V3'), c('BinaryTarget','MultiClassTarget'))
data.table::setcolorder(data, c(1, c(ncol(data)-1,ncol(data),2:(ncol(data)-2))))

# Load to warehouse
AutoQuant::PostGRE_RemoveCreateAppend(
  data = data,
  Append = TRUE,
  TableName = "App_QA_BigData",
  CloseConnection = TRUE,
  CreateSchema = NULL,
  Host = "localhost",
  DBName = "AutoQuant",
  User = "postgres",
  Port = 5432,
  Password = "",
  Temporary = FALSE,
  Connection = NULL)

## 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: `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)
```

HeatMapPlot *HeatMapPlot*

Description

Create heat maps with numeric or categorical dt

Usage

```
HeatMapPlot(
  dt,
  x = NULL,
  y = NULL,
  z = NULL,
  AggMethod = "mean",
  PercentileBuckets_X = 0.1,
  PercentileBuckets_Y = 0.1,
  NLevels_X = 33,
  NLevels_Y = 33
)
```

Arguments

dt	Source data.table
x	X-Axis variable
y	Y-Axis variable
z	Z-Axis variable
AggMethod	'mean', 'median', 'sum', 'sd', 'count'
PercentileBuckets_X	= 0.10

PercentileBuckets_Y
= 0.10
NLevels_X = 20
NLevels_Y = 20
RankLevels_X = 'mean'

Author(s)

Adrian Antico

See Also

Other Graphics: AddFacet(), BarPlot(), BoxPlot(), ChartTheme(), CorrMatrixPlot(), DensityPlot(), HistPlot(), PlotlyConversion(), StockData(), StockPlot(), ViolinPlot(), multiplot()

HistPlot *HistPlot*

Description

Build a histogram plot by simply passing arguments to a single function. It will sample your data using SampleSize number of rows. Sampled data is randomized.

Usage

```
HistPlot(
  data = NULL,
  XVar = NULL,
  YVar = NULL,
  ColorVar = NULL,
  FacetVar1 = NULL,
  FacetVar2 = NULL,
  SampleSize = 1000000L,
  Bins = 30,
  FillColor = "gray",
  OutlierSize = 0.1,
  OutlierColor = "blue",
  YTicks = "Default",
  XTicks = "Default",
  TextSize = 12,
  AngleX = 90,
  AngleY = 0,
  ChartColor = "aliceblue",
  BorderColor = "darkblue",
  TextColor = "darkblue",
  GridColor = "#d3d3e0",
  BackGroundColor = "gray95",
  SubTitleColor = "blue",
  LegendPosition = "bottom",
  LegendBorderSize = 0.5,
  LegendLineType = "solid",
  Debug = FALSE
)
```

Arguments

`data`	Source data.table
`XVar`	Column name of X-Axis variable. If NULL then ignored
`YVar`	Column name of Y-Axis variable. If NULL then ignored
`ColorVar`	Column name of Group Variable for distinct colored histograms by group levels
`FacetVar1`	Column name of facet variable 1. If NULL then ignored
`FacetVar2`	Column name of facet variable 2. If NULL then ignored
`SampleSize`	An integer for the number of rows to use. Sampled data is randomized. If NULL then ignored
`Bins`	= 30
`FillColor`	'gray'
`OutlierSize`	0.10
`OutlierColor`	'blue'
`YTicks`	Choose from 'Default', 'Percentiles', 'Every 5th percentile', 'Deciles', 'Quantiles', 'Quartiles'
`XTicks`	Choose from 'Default', '1 year', '1 day', '3 day', '1 week', '2 week', '1 month', '3 month', '6 month', '2 year', '5 year', '10 year', '1 minute', '15 minutes', '30 minutes', '1 hour', '3 hour', '6 hour', '12 hour'
`TextSize`	14
`AngleX`	90
`AngleY`	0
`ChartColor`	'lightsteelblue'
`BorderColor`	'darkblue'
`TextColor`	'darkblue'
`GridColor`	'white'
`BackGroundColor`	'gray95'
`SubTitleColor`	'darkblue'
`LegendPosition`	'bottom'
`LegendBorderSize`	0.50
`LegendLineType`	'solid'
`Debug`	FALSE

Author(s)

Adrian Antico

See Also

Other Graphics: `AddFacet()`, `BarPlot()`, `BoxPlot()`, `ChartTheme()`, `CorrMatrixPlot()`, `DensityPlot()`, `HeatMapPlot()`, `PlotlyConversion()`, `StockData()`, `StockPlot()`, `ViolinPlot()`, `multiplot()`

Examples

```
## Not run:
# Load packages
library(AutoQuant)
library(data.table)

# Load data
data <- data.table::fread(file = file.path('C:/Users/Bizon/Documents/GitHub/BenchmarkData1.csv'))

# Run function
p1 <- AutoQuant:::HistPlot(
  data = data,
  XVar = NULL,
  YVar = 'Weekly_Sales',
  ColorVar = 'Region',
  FacetVar1 = 'Store',
  FacetVar2 = 'Dept',
  SampleSize = 1000000L,
  Bins = 20,
  FillColor = 'gray',
  YTicks = 'Default',
  XTicks = 'Default',
  TextSize = 12,
  AngleX = 90,
  AngleY = 0,
  ChartColor = 'lightsteelblue1',
  BorderColor = 'darkblue',
  TextColor = 'darkblue',
  GridColor = 'white',
  BackGroundColor = 'gray95',
  SubTitleColor = 'blue',
  LegendPosition = 'bottom',
  LegendBorderSize = 0.50,
  LegendLineType = 'solid',
  Debug = FALSE)

# Step through function
# # plotly::ggplotly(p1)
# XVar = NULL
# YVar = 'Weekly_Sales'
# AggMethod = 'mean'
# ColorVar = 'Region'
# FacetVar1 = NULL
# FacetVar2 = NULL
# Bins = 20
# SampleSize = 1000000L
# FillColor = 'gray'
# YTicks = 'Default'
# XTicks = 'Default'
# TextSize = 12
# AngleX = 90
# AngleY = 0
# ChartColor = 'lightsteelblue1'
# BorderColor = 'darkblue'
# TextColor = 'darkblue'
# GridColor = 'white'
```

```
# BackGroundColor = 'gray95'
# SubTitleColor = 'blue'
# LegendPosition = 'bottom'
# LegendBorderSize = 0.50
# LegendLineType = 'solid'
# Debug = FALSE
# Bins

## End(Not run)
```

`ModelInsightsReport` *ModelInsightsReport*

Description

ModelInsightsReport is an Rmarkdown report for viewing the model insights generated by AutoQuant 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,
  ModelObject = 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 AutoQuant 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 AutoQuant supervised learning function
`Algo`	'catboost' or 'other'. Use 'catboost' if using AutoQuant::AutoCatBoost_() functions. Otherwise, 'other'
`SourcePath`	Path to directory with AutoQuant Model Output
`OutputPath`	Path to directory where the html will be saved
`ModelObject`	Returned output from regression, classificaiton, and multiclass Remix Auto_() models. Currenly supports CatBoost, XGBoost, and LightGBM models
`Test_Importance_dt`	NULL.. Ignore if using AutoQuant Models. Otherwise, supply a two column data.table with colnames 'Variable' and 'Importance'
`Validation_Importance_dt`	NULL.. Ignore if using AutoQuant Models. Otherwise, supply a two column data.table with colnames 'Variable' and 'Importance'
`Train_Importance_dt`	NULL.. Ignore if using AutoQuant Models. Otherwise, supply a two column data.table with colnames 'Variable' and 'Importance'
`Test_Interaction_dt`	NULL.. Ignore if using AutoQuant Models. Otherwise, supply a three column data.table with colnames 'Features1', 'Features2' and 'score'
`Validation_Interaction_dt`	NULL.. Ignore if using AutoQuant Models. Otherwise, supply a three column data.table with colnames 'Features1', 'Features2' and 'score'
`Train_Interaction_dt`	NULL.. Ignore if using AutoQuant 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 ModelObject is left NULL

Author(s)

Adrian Antico

See Also

Other Model Insights: `ShapImportancePlot()`

Examples

```
## Not run:

######################################################
# CatBoost
######################################################

# Create some dummy correlated data
data <- AutoQuant::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
ModelObject <- AutoQuant::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
AutoQuant::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(),
    ModelObject = ModelObject)

## 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: AddFacet(), BarPlot(), BoxPlot(), ChartTheme(), CorrMatrixPlot(), DensityPlot(), HeatMapPlot(), HistPlot(), PlotlyConversion(), StockData(), StockPlot(), ViolinPlot()

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 <- AutoQuant::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 <- AutoQuant::ParDepCalPlots(
  data,
  PredictionColName = "Predict",
  TargetColName = "Target",
  IndepVar = "Independent_Variable1",
  GraphType = "boxplot",
  PercentileBucket = 0.20,
  FactLevels = 10,
  Function = function(x) mean(x, na.rm = TRUE))
AutoQuant::multiplot(plotlist = list(p1,p2))

## End(Not run)
```

`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 = NULL,
  TargetColName = NULL,
  IndepVar = NULL,
  GraphType = "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 <- AutoQuant::FakeDataGenerator(
  Correlation = 0.70, N = 10000000, Classification = FALSE)
data.table::setnames(data, "Independent_Variable2", "Predict")

# Build plot
Plot <- AutoQuant::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)

# Step through function
# 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)
```

PlotGUI *PlotGUI*

Description

Spin up the esquisse plotting gui

Usage

```
PlotGUI()
```

See Also

Other EDA: AutoWordFreq(), EDA_Histograms(), ScatterCopula(), UserBaseEvolution()

PosteGRE_CreateDatabase
PostGRE_CreateDatabase

Description

PostGRE_CreateDatabase will create a database with a name supplied by user

Usage

```
PosteGRE_CreateDatabase(
  DBName = NULL,
  Connection = NULL,
  CloseConnection = TRUE,
  Host = "localhost",
  Port = 5432,
  User = "postgres",
  Password = ""
)
```

Arguments

DBName	See args from related functions
Connection	See args from related functions
CloseConnection	See args from related functions
Host	See args from related functions
Port	See args from related functions
User	See args from related functions
Password	See args from related functions

Author(s)

Adrian Antico

See Also

Other Database: AutoDataDictionaries(), PostGRE_AppendData(), PostGRE_CreateTable(), PostGRE_GetTableNames(), PostGRE_ListTables(), PostGRE_Query(), PostGRE_RemoveCreateAppend(), PostGRE_RemoveTable(), PosteGRE_DropDB(), PosteGRE_ListDatabases(), SQL_ClearTable(), SQL_DropTable(), SQL_Query_Push(), SQL_Query(), SQL_SaveTable(), SQL_Server_DBConnection()

PosteGRE_DropDB *PosteGRE_DropDB*

Description

PosteGRE_DropDB Drop selected database if it exists

Usage

```
PosteGRE_DropDB(
  DBName = NULL,
  Host = "localhost",
  Port = 5432,
  User = "postgres",
  Password = "",
  Connection = NULL,
  CloseConnection = TRUE
)
```

Arguments

DBName	name of db
Host	See args from related functions
Port	See args from related functions
User	See args from related functions
Password	See args from related functions
Connection	See args from related functions
CloseConnection	See args from related functions

Author(s)

Adrian Antico

See Also

Other Database: AutoDataDictionaries(), PostGRE_AppendData(), PostGRE_CreateTable(), PostGRE_GetTableNames(), PostGRE_ListTables(), PostGRE_Query(), PostGRE_RemoveCreateAppend(), PostGRE_RemoveTable(), PosteGRE_CreateDatabase(), PosteGRE_ListDatabases(), SQL_ClearTable(), SQL_DropTable(), SQL_Query_Push(), SQL_Query(), SQL_SaveTable(), SQL_Server_DBConnection()

PosteGRE_ListDatabases

PosteGRE_ListDatabases

Description

PosteGRE_ListDatabases list of available databases

Usage

```
PosteGRE_ListDatabases(
  Host = "localhost",
  Port = 5432,
  User = "postgres",
  Password = "",
  Connection = NULL,
  CloseConnection = TRUE
)
```

Arguments

Host	See args from related functions
Port	See args from related functions
User	See args from related functions
Password	See args from related functions
Connection	See args from related functions
CloseConnection	See args from related functions

Author(s)

Adrian Antico

See Also

Other Database: AutoDataDictionaries(), PostGRE_AppendData(), PostGRE_CreateTable(), PostGRE_GetTableNames(), PostGRE_ListTables(), PostGRE_Query(), PostGRE_RemoveCreateAppend(), PostGRE_RemoveTable(), PosteGRE_CreateDatabase(), PosteGRE_DropDB(), SQL_ClearTable(), SQL_DropTable(), SQL_Query_Push(), SQL_Query(), SQL_SaveTable(), SQL_Server_DBConnection()

PostGRE_AppendData *PostGRE_AppendData*

Description

PostGRE_AppendData get data from a database table

Usage

```
PostGRE_AppendData(
  data = NULL,
  TableName = NULL,
  Append = FALSE,
  Connection = NULL,
  CloseConnection = FALSE,
  Host = NULL,
  DBName = NULL,
  User = NULL,
  Port = NULL,
  Password = NULL
)
```

Arguments

data	Source data.table
Append	Set to TRUE to append data, FALSE to overwrite data
Connection	db connection
CloseConnection	= FALSE
Host	If Connection is NULL then this must be supplied. host
DBName	If Connection is NULL then this must be supplied. dbname
User	If Connection is NULL then this must be supplied. user
Port	If Connection is NULL then this must be supplied. port
Password	If Connection is NULL then this must be supplied. password

Author(s)

Adrian Antico

See Also

Other Database: AutoDataDictionaries(), PostGRE_CreateTable(), PostGRE_GetTableNames(), PostGRE_ListTables(), PostGRE_Query(), PostGRE_RemoveCreateAppend(), PostGRE_RemoveTable(), PosteGRE_CreateDatabase(), PosteGRE_DropDB(), PosteGRE_ListDatabases(), SQL_ClearTable(), SQL_DropTable(), SQL_Query_Push(), SQL_Query(), SQL_SaveTable(), SQL_Server_DBConnection()

Examples

```
## Not run:
AutoQuant::PostGRE_AppendData(
  data = data,
  TableName = 'somename',
  Append = FALSE,
  CloseConnection = FALSE,
  Host = 'localhost',
  DBName = 'AutoQuant',
  User = 'postgres',
  Port = 5432,
  Password = 'Aa...')

# data = data
# CloseConnection = FALSE,
# Host = 'localhost'
# DBName = 'Testing'
# User = 'postgres'
# Port = 5432
# Password = 'Aa...'

## End(Not run)
```

PostGRE_CreateTable *PostGRE_CreateTable*

Description

PostGRE_CreateTable get data from a database table

Usage

```
PostGRE_CreateTable(
  data = NULL,
  DBName = NULL,
  Schema = NULL,
  TableName = NULL,
  Connection = NULL,
  CloseConnection = FALSE,
  Temporary = FALSE,
  Host = NULL,
  User = NULL,
  Port = NULL,
  Password = NULL
)
```

Arguments

data	Source data.table. If you supply a Schema, data will be ignored.
DBName	If Connection is NULL then this must be supplied. database name
Schema	Optional. Advised to use if type inference is fuzzy

`TableName`	Name of table you want created
`Connection`	NULL. If supplied, use this: Connection <- DBI::dbConnect(RPostgres::Postgres(), host = Host, dbname = DBName, user = User, port = Port, password = Password)
`CloseConnection`	= FALSE
`Temporary`	If Connection is NULL then this must be supplied. FALSE
`Host`	If Connection is NULL then this must be supplied. host name
`User`	If Connection is NULL then this must be supplied. user name
`Port`	If Connection is NULL then this must be supplied. port name
`Password`	user password

Author(s)

Adrian Antico

See Also

Other Database: `AutoDataDictionaries()`, `PostGRE_AppendData()`, `PostGRE_GetTableNames()`, `PostGRE_ListTables()`, `PostGRE_Query()`, `PostGRE_RemoveCreateAppend()`, `PostGRE_RemoveTable()`, `PosteGRE_CreateDatabase()`, `PosteGRE_DropDB()`, `PosteGRE_ListDatabases()`, `SQL_ClearTable()`, `SQL_DropTable()`, `SQL_Query_Push()`, `SQL_Query()`, `SQL_SaveTable()`, `SQL_Server_DBConnection()`

Examples

```
## Not run:
AutoQuant::PostGRE_CreateTable(
  data,
  DBName = 'Testing',
  Schema = NULL,
  TableName = NULL,
  Temporary = FALSE,
  Connection = NULL,
  CloseConnection = FALSE,
  Host = 'localhost',
  User = 'postgres',
  Port = 5432,
  Password = 'Aa...')

## End(Not run)
```

`PostGRE_GetTableNames` *PostGRE_GetTableNames*

Description

PostGRE_GetTableNames will list all column names from a table

Usage

```
PostGRE_GetTableNames(
  Host = NULL,
  CloseConnection = FALSE,
  DBName = NULL,
  TableName = NULL,
  User = NULL,
  Port = NULL,
  Password = NULL
)
```

Arguments

`Host`	See args from related functions
`CloseConnection`	See args from related functions
`DBName`	See args from related functions
`TableName`	Name of postgres table
`User`	See args from related functions
`Port`	See args from related functions
`Password`	See args from related functions

Value

A character vector of names. Exactly like names R base function for a data.frame

Author(s)

Adrian Antico

See Also

Other Database: `AutoDataDictionaries()`, `PostGRE_AppendData()`, `PostGRE_CreateTable()`, `PostGRE_ListTables()`, `PostGRE_Query()`, `PostGRE_RemoveCreateAppend()`, `PostGRE_RemoveTable()`, `PosteGRE_CreateDatabase()`, `PosteGRE_DropDB()`, `PosteGRE_ListDatabases()`, `SQL_ClearTable()`, `SQL_DropTable()`, `SQL_Query_Push()`, `SQL_Query()`, `SQL_SaveTable()`, `SQL_Server_DBConnection()`

`PostGRE_ListTables` *PostGRE_ListTables*

Description

PostGRE_ListTables will list all tables with an associated db

Usage

```
PostGRE_ListTables(
  DBName = NULL,
  Connection = NULL,
  CloseConnection = TRUE,
  Host = NULL,
  Port = NULL,
  User = NULL,
  Password = NULL
)
```

Arguments

DBName	See args from related functions
Connection	See args from related functions
CloseConnection	See args from related functions
Host	See args from related functions
Port	See args from related functions
User	See args from related functions
Password	See args from related functions
Temporary	See args from related functions

Author(s)

Adrian Antico

See Also

Other Database: AutoDataDictionaries(), PostGRE_AppendData(), PostGRE_CreateTable(), PostGRE_GetTableNames(), PostGRE_Query(), PostGRE_RemoveCreateAppend(), PostGRE_RemoveTable(), PosteGRE_CreateDatabase(), PosteGRE_DropDB(), PosteGRE_ListDatabases(), SQL_ClearTable(), SQL_DropTable(), SQL_Query_Push(), SQL_Query(), SQL_SaveTable(), SQL_Server_DBConnection()

PostGRE_Query *PostGRE_Query*

Description

PostGRE_Query get data from a database table

Usage

```
PostGRE_Query(
  Query = NULL,
  Connection = NULL,
  CloseConnection = FALSE,
  Host = NULL,
  DBName = NULL,
```

```
  User = NULL,
  Port = NULL,
  Password = NULL
)
```

Arguments

Query	SQL Statement in quotes
Connection	db connection
CloseConnection	= FALSE
Host	If Connection is NULL then this must be supplied. host
DBName	If Connection is NULL then this must be supplied. dbname
User	If Connection is NULL then this must be supplied. user
Port	If Connection is NULL then this must be supplied. port
Password	If Connection is NULL then this must be supplied. password

Author(s)

Adrian Antico

See Also

Other Database: AutoDataDictionaries(), PostGRE_AppendData(), PostGRE_CreateTable(), PostGRE_GetTableNames(), PostGRE_ListTables(), PostGRE_RemoveCreateAppend(), PostGRE_RemoveTable(), PosteGRE_CreateDatabase(), PosteGRE_DropDB(), PosteGRE_ListDatabases(), SQL_ClearTable(), SQL_DropTable(), SQL_Query_Push(), SQL_Query(), SQL_SaveTable(), SQL_Server_DBConnection()

Examples

```
## Not run:

# Query data from table with Uppercase name
data <- AutoQuant::PostGRE_Query(
  Query = paste0("SELECT * FROM ", shQuote('Devices')),
  Host = 'localhost',
  CloseConnection = FALSE,
  DBName = 'Testing',
  User = 'postgres',
  Port = 5432,
  Password = 'Aa...')

# Query = 'Select * from static_data'
# Host = 'localhost'
# DBName = 'Testing'
# CloseConnection = FALSE,
# User = 'postgres'
# Port = 5432
# Password = 'Aa...'

# Create Schema
query <- "CREATE SCHEMA AutoQuant AUTHORIZATION postgres;"
AutoQuant::PostGRE_Query(
```

```
  Query = query,
  Host = 'localhost',
  CloseConnection = FALSE,
  DBName = 'Testing',
  User = 'postgres',
  Port = 5432,
  Password = 'Aa...')

## End(Not run)
```

`PostGRE_RemoveCreateAppend` *PostGRE_RemoveCreateAppend*

Description

PostGRE_RemoveCreateAppend will DROP the table specified

Usage

```
PostGRE_RemoveCreateAppend(
  data = NULL,
  TableName = NULL,
  CloseConnection = TRUE,
  CreateSchema = NULL,
  Host = NULL,
  DBName = NULL,
  User = NULL,
  Port = NULL,
  Password = NULL,
  Temporary = FALSE,
  Connection = NULL,
  Append = TRUE
)
```

Arguments

`data`	See args from related functions
`TableName`	See args from related functions
`CloseConnection`	See args from related functions
`CreateSchema`	See args from related functions
`Host`	See args from related functions
`DBName`	See args from related functions
`User`	See args from related functions
`Port`	See args from related functions
`Password`	See args from related functions
`Temporary`	See args from related functions
`Connection`	See args from related functions
`Append`	See args from related functions

Author(s)

Adrian Antico

See Also

Other Database: AutoDataDictionaries(), PostGRE_AppendData(), PostGRE_CreateTable(), PostGRE_GetTableNames(), PostGRE_ListTables(), PostGRE_Query(), PostGRE_RemoveTable(), PosteGRE_CreateDatabase(), PosteGRE_DropDB(), PosteGRE_ListDatabases(), SQL_ClearTable(), SQL_DropTable(), SQL_Query_Push(), SQL_Query(), SQL_SaveTable(), SQL_Server_DBConnection()

PostGRE_RemoveTable *PostGRE_RemoveTable*

Description

PostGRE_RemoveTable will DROP the table specified

Usage

```
PostGRE_RemoveTable(
  TableName = NULL,
  Connection = NULL,
  CloseConnection = FALSE,
  Host = NULL,
  DBName = NULL,
  User = NULL,
  Port = NULL,
  Password = NULL
)
```

Arguments

TableName	Name of table you want created
Connection	NULL. If supplied, use this: Connection <- DBI::dbConnect(RPostgres::Postgres(), host = Host, dbname = DBName, user = User, port = Port, password = Password)
CloseConnection	= FALSE
Host	If Connection is NULL then this must be supplied. Host name
DBName	If Connection is NULL then this must be supplied. database name
User	If Connection is NULL then this must be supplied. user name
Port	If Connection is NULL then this must be supplied. port name
Password	If Connection is NULL then this must be supplied. user password

Author(s)

Adrian Antico

See Also

Other Database: AutoDataDictionaries(), PostGRE_AppendData(), PostGRE_CreateTable(), PostGRE_GetTableNames(), PostGRE_ListTables(), PostGRE_Query(), PostGRE_RemoveCreateAppend(), PosteGRE_CreateDatabase(), PosteGRE_DropDB(), PosteGRE_ListDatabases(), SQL_ClearTable(), SQL_DropTable(), SQL_Query_Push(), SQL_Query(), SQL_SaveTable(), SQL_Server_DBConnection()

Examples

```
## Not run:
AutoQuant::PostGRE_RemoveTable(
  TableName = 'static_data',
  Connection = NULL,
  CloseConnection = FALSE,
  Host = 'localhost',
  DBName = 'Testing',
  User = 'postgres',
  Port = 5432,
  Password = 'Aa...')

# Host = 'localhost'
# TableName = 'static_data'
# Connection = NULL
# DBName = 'Testing'
# User = 'postgres'
# Port = 5432
# Password = 'Aa...'

## End(Not run)
```

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: GenTSAnomVars()

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 <- AutoQuant::FakeDataGenerator(
  Correlation = 0.80,
  N = 250000,
  ID = 0,
  FactorCount = 0,
  AddDate = TRUE,
```

```
    AddComment = FALSE,
    AddWeightsColumn = FALSE,
    ZIP = 0)

  # Build Plots
  output <- AutoQuant::ResidualPlots(
    TestData = test_data,
    Target = "Adrian",
    Predicted = "Independent_Variable1",
    DateColumnName = "DateTime",
    Gam_Fit = TRUE)

  ## 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()

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,
  Marginals = FALSE,
  MarginalType = "density",
  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",
  Debug = FALSE
)
```

Arguments

data	Source data.table
x_var	Numeric variable
y_var	Numeric variable
Marginals	= FALSE,
MarginalType	= 'density',
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
Debug	= FALSE

Author(s)

Adrian Antico

See Also

Other EDA: AutoWordFreq(), EDA_Histograms(), PlotGUI(), UserBaseEvolution()

Examples

```
## Not run:
# Create data
data <- AutoQuant::FakeDataGenerator()

# Build plot
AutoQuant::ScatterCopula(
  data = data,
  x_var = 'Independent_Variable1',
  y_var = 'Independent_Variable2',
  Marginals = FALSE,
  MarginalType = 'density',
  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",
  Debug = FALSE)

## End(Not run)
```

ShapImportancePlot *ShapImportancePlot*

Description

Generate Variable Importance Plots using Shapely Values of given data set

Usage

```
ShapImportancePlot(
  data,
  ShapColNames = NULL,
  FacetVar1 = NULL,
  FacetVar2 = NULL,
  AggMethod = "mean",
  TopN = 25,
  Debug = FALSE
)
```

Arguments

data	Source data.table
ShapColNames	Names of the columns that contain shap values you want included
FacetVar1	Column name
FacetVar2	Column name
AggMethod	A string for aggregating shapely values for importances. Choices include, 'mean', 'absmean', 'meanabs', 'sd', 'median', 'absmedian', 'medianabs'
TopN	The number of variables to plot
Debug	= FALSE

Author(s)

Adrian Antico

See Also

Other Model Insights: ModelInsightsReport()

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	AutoQuant::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(), PostGRE_AppendData(), PostGRE_CreateTable(), PostGRE_GetTableNames(), PostGRE_ListTables(), PostGRE_Query(), PostGRE_RemoveCreateAppend(), PostGRE_RemoveTable(), PosteGRE_CreateDatabase(), PosteGRE_DropDB(), PosteGRE_ListDatabases(), 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	AutoQuant::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(), PostGRE_AppendData(), PostGRE_CreateTable(), PostGRE_GetTableNames(), PostGRE_ListTables(), PostGRE_Query(), PostGRE_RemoveCreateAppend(), PostGRE_RemoveTable(), PosteGRE_CreateDatabase(), PosteGRE_DropDB(), PosteGRE_ListDatabases(), 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	AutoQuant::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(), PostGRE_AppendData(), PostGRE_CreateTable(), PostGRE_GetTableNames(), PostGRE_ListTables(), PostGRE_Query(), PostGRE_RemoveCreateAppend(), PostGRE_RemoveTable(), PosteGRE_CreateDatabase(), PosteGRE_DropDB(), PosteGRE_ListDatabases(), 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	AutoQuant::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(), PostGRE_AppendData(), PostGRE_CreateTable(), PostGRE_GetTableNames(), PostGRE_ListTables(), PostGRE_Query(), PostGRE_RemoveCreateAppend(), PostGRE_RemoveTable(), PosteGRE_CreateDatabase(), PosteGRE_DropDB(), PosteGRE_ListDatabases(), 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	AutoQuant::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(), PostGRE_AppendData(), PostGRE_CreateTable(), PostGRE_GetTableNames(), PostGRE_ListTables(), PostGRE_Query(), PostGRE_RemoveCreateAppend(), PostGRE_RemoveTable(), PosteGRE_CreateDatabase(), PosteGRE_DropDB(), PosteGRE_ListDatabases(), 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(), PostGRE_AppendData(), PostGRE_CreateTable(), PostGRE_GetTableNames(), PostGRE_ListTables(), PostGRE_Query(), PostGRE_RemoveCreateAppend(), PostGRE_RemoveTable(), PosteGRE_CreateDatabase(), PosteGRE_DropDB(), PosteGRE_ListDatabases(), SQL_ClearTable(), SQL_DropTable(), SQL_Query_Push(), SQL_Query(), SQL_SaveTable()

StockData *StockData*

Description

Create stock data for plotting using StockPlot()

Usage

```
StockData(
  PolyOut = NULL,
  Symbol = "TSLA",
  CompanyName = "Tesla Inc. Common Stock",
  Metric = "Stock Price",
  TimeAgg = "days",
  StartDate = "2022-01-01",
  EndDate = "2022-01-01",
  APIKey = NULL
)
```

Arguments

PolyOut	NULL. If NULL, data is pulled. If supplied, data is not pulled.
Symbol	ticker symbol string
CompanyName	company name if you have it. ends up in title, that is all
Metric	Stock Price, Percent Returns (use symbol for percent), Percent Log Returns (use symbol for percent), Index, Quadratic Variation
TimeAgg	= 'days', 'weeks', 'months'
StartDate	Supply a start date. E.g. '2022-01-01'
EndDate	Supply an end date. E.g. 'Sys.Date()'
APIKey	Supply your polygon API key
Type	'candlestick', 'ohlc'

Author(s)

Adrian Antico

See Also

Other Graphics: AddFacet(), BarPlot(), BoxPlot(), ChartTheme(), CorrMatrixPlot(), DensityPlot(), HeatMapPlot(), HistPlot(), PlotlyConversion(), StockPlot(), ViolinPlot(), multiplot()

`StockPlot` *StockPlot*

Description

Create a candlestick plot for stocks. See https://plotly.com/r/figure-labels/

Usage

```
StockPlot(StockDataOutput, Type = "candlestick")
```

Arguments

`StockDataOutput`	PolyOut returned from StockData()
`Type`	'candlestick', 'ohlc'

Author(s)

Adrian Antico

See Also

Other Graphics: `AddFacet()`, `BarPlot()`, `BoxPlot()`, `ChartTheme()`, `CorrMatrixPlot()`, `DensityPlot()`, `HeatMapPlot()`, `HistPlot()`, `PlotlyConversion()`, `StockData()`, `ViolinPlot()`, `multiplot()`

`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)
```

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()`, `PlotGUI()`, `ScatterCopula()`

ViolinPlot *ViolinPlot*

Description

Build a violin plot by simply passing arguments to a single function. It will sample your data using SampleSize number of rows. Sampled data is randomized.

Usage

```
ViolinPlot(
  data = NULL,
  XVar = NULL,
  YVar = NULL,
  FacetVar1 = NULL,
  FacetVar2 = NULL,
  SampleSize = 1000000L,
  FillColor = "gray",
  YTicks = "Default",
  XTicks = "Default",
  TextSize = 12,
  AngleX = 90,
  AngleY = 0,
  ChartColor = "lightsteelblue1",
  BorderColor = "darkblue",
  TextColor = "darkblue",
  GridColor = "white",
  BackGroundColor = "gray95",
  SubTitleColor = "blue",
  LegendPosition = "bottom",
  LegendBorderSize = 0.5,
  LegendLineType = "solid",
  Debug = FALSE
)
```

Arguments

data	Source data.table
XVar	Column name of X-Axis variable. If NULL then ignored
YVar	Column name of Y-Axis variable. If NULL then ignored
FacetVar1	Column name of facet variable 1. If NULL then ignored
FacetVar2	Column name of facet variable 2. If NULL then ignored
SampleSize	An integer for the number of rows to use. Sampled data is randomized. If NULL then ignored
FillColor	'gray'
YTicks	Choose from 'Default', 'Percentiles', 'Every 5th percentile', 'Deciles', 'Quantiles', 'Quartiles'

XTicks	Choose from 'Default', '1 year', '1 day', '3 day', '1 week', '2 week', '1 month', '3 month', '6 month', '2 year', '5 year', '10 year', '1 minute', '15 minutes', '30 minutes', '1 hour', '3 hour', '6 hour', '12 hour'
TextSize	14
AngleX	90
AngleY	0
ChartColor	'lightsteelblue'
BorderColor	'darkblue'
TextColor	'darkblue'
GridColor	'white'
BackGroundColor	'gray95'
SubTitleColor	'darkblue'
LegendPosition	'bottom'
LegendBorderSize	0.50
LegendLineType	'solid'
Debug	FALSE

Author(s)

Adrian Antico

See Also

Other Graphics: AddFacet(), BarPlot(), BoxPlot(), ChartTheme(), CorrMatrixPlot(), DensityPlot(), HeatMapPlot(), HistPlot(), PlotlyConversion(), StockData(), StockPlot(), multiplot()

Examples

```
## Not run:
# Load packages
library(AutoQuant)
library(data.table)

# Load data
data <- data.table::fread(file = file.path('C:/Users/Bizon/Documents/GitHub/BenchmarkData1.csv'))

# Run function
AutoQuant:::ViolinPlot(
  data = data,
  XVar = 'Region',
  YVar = 'Weekly_Sales',
  FacetVar1 = 'Store',
  FacetVar2 = NULL,
  SampleSize = 1000000L,
  FillColor = 'gray',
  YTicks = 'Default',
  XTicks = 'Default',
  TextSize = 12,
  AngleX = 90,
```

```
  AngleY = 0,
  ChartColor = 'lightsteelblue1',
  BorderColor = 'darkblue',
  TextColor = 'darkblue',
  GridColor = 'white',
  BackGroundColor = 'gray95',
  SubTitleColor = 'blue',
  LegendPosition = 'bottom',
  LegendBorderSize = 0.50,
  LegendLineType = 'solid',
  Debug = FALSE)

# Step through function
# XVar = 'Region'
# YVar = 'Weekly_Sales'
# FacetVar1 = 'Store'
# FacetVar2 = NULL
# SampleSize = 1000000L
# FillColor = 'gray'
# YTicks = 'Default'
# XTicks = 'Default'
# TextSize = 12
# AngleX = 90
# AngleY = 0
# ChartColor = 'lightsteelblue1'
# BorderColor = 'darkblue'
# TextColor = 'darkblue'
# GridColor = 'white'
# BackGroundColor = 'gray95'
# SubTitleColor = 'blue'
# LegendPosition = 'bottom'
# LegendBorderSize = 0.50
# LegendLineType = 'solid'
# Debug = FALSE

## End(Not run)
```

Index