Package ‘DriveML’

June 4, 2020

Type Package
Title Self-Drive Machine Learning Projects
Version 0.1.0
Maintainer Dayanand Ubrangala <daya6489@gmail.com>
Depends R (>= 3.3.0)
Imports sampling(>= 2.8), rmarkdown(>= 1.9), data.table(>= 1.10.4-3),
SmartEDA(>= 0.3.1), caTools, ParamHelpers(>= 1.12), mlr(>=
2.15.0), ggplot2(>= 2.2.1), iml
Description Implementing some of the pillars of an automated machine learn-
ing pipeline such as (i) Automated data preparation, (ii) Feature engineering, (iii) Model build-
ing in classification context that includes techniques such as (a) Regularised regres-
sion [1], (b) Logistic regression [2], (c) Random Forest [3], (d) Decision tree [4] and (e) Ex-
treme Gradient Boosting (xgboost) [5], and finally, (iv) Model explanation (us-
License MIT + file LICENSE
Suggests testthat, knitr, ranger, glmnet, randomForest, rpart,
xgboost, stats, graphics, tidyR, MASS
Encoding UTF-8
LazyData true
Repository CRAN
RoxygenNote 6.1.1
VignetteBuilder knitr
NeedsCompilation no
autoDataprep

Automatic data preparation for ML algorithm

Description

Final data preparation before ML algorithm. Function provides final data set and highlights of the
data preparation

Usage

autoDataprep(data, target = NULL, missimpute = "default",
auto_mar = FALSE, mar_object = NULL, dummyvar = TRUE,
char_var_limit = 12, aucv = 0.02, corr = 0.99,
outlier_flag = FALSE, interaction_var = FALSE,
frequent_var = FALSE, uid = NULL, onlykeep = NULL, drop = NULL,
verbose = FALSE)
Arguments

- **data**: [data.frame | Required] dataframe or data.table
- **target**: [integer | Required] dependent variable (binary or multiclass)
- **missimpute**: [text | Optional] missing value impuation using mlr misimpute function. See more methods in details
- **auto_mar**: [character | Optional] identify any missing variable which are completely missing at random or not.(default FALSE). If TRUE this will call autoMAR()
- **mar_object**: [character | Optional] object created from autoMAR function
- **dummyvar**: [logical | Optional] categorical feature engineering i.e. one hot encoding (default is TRUE)
- **char_var_limit**: [integer | Optional] default limit is 12 for a dummy variable preparation. Ex: if gender variable has two different value "M" and "F", then gender has 2 level
- **aucv**: [integer | Optional] cut off value for AUC based variable selection
- **corr**: [integer | Optional] cut off value for correlation based variable selection
- **outlier_flag**: [logical | Optional] to add outlier features (default is False)
- **interaction_var**: [logical | Optional] bulk interactions transformer for numerical features
- **frequent_var**: [logical | Optional] Frequent transformer for categorical features
- **uid**: [character | Optional] unique identifier column if any to keep in the final data set
- **onlykeep**: [character | Optional] only consider selected variables for data preparation
- **drop**: [character | Optional] exclude variable list from the data preparation
- **verbose**: [logical | Optional] display executions steps on console. Default FALSE

Details

Missing imputation using impute function from MLR
MLR package have a appropriate way to impute missing value using multiple methods. default value is listed below #'

- mean value for integer variable
- median value for numeric variable
- mode value for character or factor variable

Optional: You might be interested to impute missing variable using ML method. List of algortihms will be handle missing variables in MLR package listLearners("classif", check.packages = TRUE, properties = "missings")[c("class", "package")]

Feature engineering

- Missing not completely at random variable using autoMAR function
- Date transformer like year, month, quarter, week
- Frequent transformer counts each categorical value in the dataset
- Interaction transformer using multiplication
autoDataprep

• one hot dummy coding for categorical value
• outlier flag and capping variable for numerical value

Feature reduction

• Zero variance using nearZeroVar caret function
• Pearson’s Correlation value
• AUC with target variable

Value

list output contains below objects

complete_data Complete data set including new novel features based on the functional understanding of the dataset
master_data filtered data set based on the input parameter
final_var_list list of master variables
auc_var list of auc variables
cor_var list of correlation variables
overall_var all variables in the dataset
zerovariance zero variance variables in the dataset

See Also

impute

Examples

#Auto data prep
traindata <- autoDataprep(heart, target = "target_var", missimpute = "default",
dummyvar = TRUE, aucv = 0.02, corr = 0.98, outlier_flag = TRUE,
interaction_var = TRUE, frequent_var = TRUE)
train <- traindata$master

# Print auto data prep object
print(autoDataprep(traindata))
**autoMAR**

*Function to identify and generate the Missing at Random features (MAR)*

---

### Description

this function will automatically identify the missing pattern and flag the variable if they are not missing at random based on AUC method

### Usage

```r
autoMAR(data, aucv = 0.9, strataname = NULL, stratasize = NULL, mar_method = "glm")
```

### Arguments

- `data`: dataframe or data.table
- `aucv`: AUC cut-off value for the not missing at random variable selection
- `strataname`: vector of stratification variables
- `stratasize`: vector of stratum sample sizes (in the order in which the strata are given in the input data set).
- `mar_method`: missing at random classification method ("glm", "rf"). Default GLM is used (GLM is run faster for high dimension data)

### Value

List output including missing variable summary and number of MAR flag variables

### Examples

```r
# Create missing at random features
marobj <- autoMAR(heart, aucv = 0.9, strataname = NULL, stratasize = NULL, mar_method = "glm")
```

---

**autoMLmodel**

*Automated machine learning training of models*

---

### Description

Automated training, tuning and validation of machine learning models. Models are tuned and re-sampling validated on an experiment set and trained on the full set and validated and testing on external sets. Classification models tune the probability threshold automatically and returns the results. Each model contains information of performance, the trained model as well as some plots.
autoMLmodel(train, test = NULL, score = NULL, target = NULL, 
testSplit = 0.2, tuneIters = 200, tuneType = "random", 
models = "all", perMetric = "auc", varImp = 20, liftGroup = 50, 
maxObs = 10000, uid = NULL, pdp = FALSE, positive = 1, 
htmlreport = FALSE, seed = 1991, verbose = FALSE)

Arguments

train [data.frame | Required] Training set

test [data.frame | Optional] Optional testing set to validate models on. If none is provided, one will be created internally. Default of NULL

score [data.frame | Optional] Optional score the models on best trained model based on AUC. If none is provided, scorelist will be null. Default of NULL

target [integer | Required] If a target is provided classification or regression models will be trained, if left as NULL unsupervised models will be trained. Default of NULL

testSplit [numeric | Optional] Percentage of data to allocate to the test set. Stratified sampling is done. Default of 0.1

tuneIters [integer | Optional] Number of tuning iterations to search for optimal hyper parameters. Default of 10

tuneType [character | Optional] Tune method applied, list of options are:
  • "random" - random search hyperparameter tuning
  • "frace" - frace uses iterated f-racing algorithm for the best solution from irace package

models [character | Optional] Which models to train. Default is all. List of strings denoting which algorithms to use for the process:
  • randomForestRandom forests using the randomForest package
  • rangerRandom forests using the ranger package
  • xgboostGradient boosting using xgboost
  • rpartdecision tree classification using rpart
  • glmnetregularised regression from glmnet
  • logreglogistic regression from stats

perMetric [character | Optional] Model validation metric. Default "auc"
  • auc - Area under the curve; mlr::auc
  • accuracy - Accuracy; mlr::acc
  • balancedAccuracy - Balanced accuracy; mlr::bac
  • brier - Brier score; mlr::brier
  • f1 - F1 measure; mlr::f1
  • meanPrecRecall - Geometric mean of precision and recall; mlr::gpr
  • logloss - Logarithmic loss; mlr::logloss

varImp [integer | Optional] Number of important features to plot
autoMLmodel

- **liftGroup**: [integer | Optional] Number of lift value to validate the test model performance
- **maxObs**: [numeric | Optional] Number of observations in the experiment training set on which models are trained, tuned and resampled on. Default of 40000. If the training set has less than 40k observations all will be used
- **uid**: [character | Optional] unique variable to keep in test output data
- **pdp**: [logical | Optional] Partial dependence plot for top important variables
- **positive**: [character | Optional] positive class for the target variable
- **htmlreport**: [logical | Optional] to view the model outcome in html format
- **seed**: [integer | Optional] Random number seed for reproducible results
- **verbose**: [logical | Optional] display executions steps on console. Default FALSE

**Details**

All the models trained using mlr train function, all of the functionality in mlr package can be applied to the autoMLmodel outcome.

autoMLmodel provides below information of the machine learning classification models:

- **trainedModels**: Model level list output contains trained model object, hyper parameters, tuned data, test data, performance and Model plots
- **results**: Summary of all trained model result like AUC, Precision, Recall, F1 score
- **modelexp**: Model gain chart
- **predicted_score**: Predicted score
- **datasummary**: Summary of the input data

**Value**

List output contains trained models and results

**See Also**

- `mlr train`
- `caret train`
- `makeLearner`
- `tuneParams`

**Examples**

```r
# Run only Logistic regression model
mymodel <- autoMLmodel(train = heart, test = NULL, target = 'target_var',
testSplit = 0.2, tuneIters = 10, tuneType = "random", models = "logreg",
varImp = 10, liftGroup = 50, maxObs = 4000, uid = NULL, seed = 1991)
```
autoMLReport

Display autoMLmodel output in HTML format using Rmarkdown

Description

This function will generate R markdown report for DriveML model object.

Usage

autoMLReport(mlobject, mldata = NULL, op_file = NULL, op_dir = NULL)

Arguments

- **mlobject**: [autoMLmodel Object | Required] autoMLmodel function output
- **mldata**: [autoDataprep Object | Optional] autoDataprep function output
- **op_file**: [character | Required] output file name (.html)
- **op_dir**: [character | Optional] output path. Default path is current working directory

Details

Using this function easily present the model outcome in standard HTML format without writing Rmarkdown scripts.

Value

HTML R Markdown output

Examples

```r
## Creating HTML report
autoMLReport(heart.model, mldata = NULL, op_file = "sample.html", op_dir = tempdir())
```

autoPDP

Generate partial dependence plots

Description

Partial dependence plots (PDPs) help you to visualize the relationship between a subset of the features and the response while accounting for the average effect of the other predictors in the model. They are particularly effective with black box models like random forests and support vector machines.
Usage

autoPDP(train, trainedModel, target, feature, sample = 0.5, modelname, seed = 1991)

Arguments

- **train**: [data.frame | Required] Training sample used to train ML model
- **trainedModel**: [model object | Required] The object holding the machine learning model and the data
- **target**: [character | Optional] Target variable name. Specify target variable if model object is other than MLR or driveML
- **feature**: [character | Optional] The feature name for which to compute the effects
- **sample**: [numeric | Optional] Percentage of sample to be considered for training set for faster computation. Default of 0.5
- **modelname**: [character | Optional] Specify which model to be plotted
- **seed**: [integer | Optional] Random seed number. Default 121

Value

List object containing a plot for each feature listed.

See Also

FeatureEffects plotPartialDependence partial

Examples

```r
# Example using DriveML model object
mymodel = heart.model
pdp_chol = autoPDP(heart, mymodel, feature = "chol", sample = 0.8, seed = 1234)

# Type one MLR package
mod <- mlr::train(makeLearner("classif.ranger"), iris.task)
cc = autoPDP(iris, mod, feature = c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width"), sample = 1, seed = 121)

# Type 2 DriveML object
hearML <- autoMLmodel(heart, target = "target_var", testSplit = 0.2, tuneIters = 10, tuneType = "random", models = "all", varImp = 20, liftGroup = 50, positive = 1, seed = 1991)
cc = autoPDP(heart, hearML, feature = "chol", sample = 0.8, seed = 1234)

cc1 = autoPDP(heart, trainedModel, target = "target_var", feature = "chol", sample = 1, modelname = "logreg", seed = 121)

# Type 3 other ML object
library(randomForest)
library(MASS)
```
```r
equation = randomForest(medv ~ ., data = Boston, ntree = 50)
c = autoPDP(Boston, eqn, target = "medv", feature = "nox", sample = 1, seed = 121)
```

**generateFeature**

Automated column transformer

**Description**

This function automatically scans through each variable and generate features based on the type listed in the detail.

**Usage**

```r
generateFeature(data, varlist, type = "Frequent", method = NULL)
```

**Arguments**

- `data`: dataframe or data.table
- `varlist`: variable list to generate the additional features
- `type`: variable transformation type 'Dummy', 'Outlier', 'Frequent', 'Interaction'
- `method`: input for variable transformation. For `type = 'Frequent'` then type should be 'Frequency' or 'Percent'. Other type method list is provided in details.

**Details**

This function is for generating features based on different transformation methods like interaction, outliers, Dummy coding etc.

**Interaction type**

- multiply - multiplication
- add - addition
- subtract - subtraction
- divide - division

**Frequency type**

- Frequency - Frequency
- Percent - Percentage

**Outlier type**

- Flag - Flag outlier values like 1 or 0
- Capping - Impute outlier value by 95th or 5th percentile value

**Date type**
• Year
• Month
• Quarter
• Week

Value

generated transformed features

Examples

# Generate interaction features
generateFeature(heart, varlist = c("cp", "chol", "trestbps"), type = "Interaction", method = "add")
generateFeature(heart, varlist = c("cp", "chol", "trestbps"), type = "Interaction", method = "multiply")

# Generate frequency features
generateFeature(heart, varlist = c("cp", "thal"), type = "Frequent", method = "Percent")
generateFeature(heart, varlist = c("cp", "thal"), type = "Frequent", method = "Frequency")

Dataset Heart Disease - Classifications

Description

This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "goal" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4.

Usage

heart

Format

A data frame with 303 rows and 14 variables:

age integer Age
sex integer Sex
cp integer chest pain type (4 values)
trestbps integer resting blood pressure
chol integer serum cholesterol in mg/dl
fbs integer fasting blood sugar > 120 mg/dl
restecg  integer resting electrocardiographic results (values 0,1,2)
thalach  integer maximum heart rate achieved
exang   integer exercise induced angina
oldpeak  double oldpeak = ST depression induced by exercise relative to rest
slope   integer the slope of the peak exercise ST segment
ca   integer number of major vessels (0-3) colored by fluoroscopy
thal   integer thal: 3 = normal; 6 = fixed defect; 7 = reversible defect
target_var integer the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4

Value
sample data

Source
https://www.kaggle.com/cdabakoglu/heart-disease-classifications-machine-learning

Examples
```r
## Load heart data
data(heart)
```

heart.model    Heart Classification Drive ML Model.

Description
Contains the task ('heart.model').

Usage
heart.model

Format
An object of class autoMLmodel of length 6.

Value
heart data driveML sample model output

References
See https://www.kaggle.com/cdabakoglu/heart-disease-classifications-machine-learning
Examples

## Sample model object
modelobj <- heart.model

misspattern  
**Missing pattern analysis for missing data**

Description

this function for summarise the missing variable, missing pattern identification, classifying the columns based on pattern of missing values.

Usage

misspattern(data, mfeature, drop = 0.99, print = FALSE)

Arguments

data: [data.frame | Required] data set with missing values
mfeature: [character | Required] only missing variable name
drop: [numeric | optional] drop variable percentage. Example, if drop = 0.9, function will automatically drop 90per missing columns from the data set
print: [character | optional] defualt print is FALSE

Value

final variable list, summary of missing data analysis

Examples

## Sample iris data
mdata <- iris
mobject <- misspattern(mdata, mfeature = c("Sepal.Length", "Petal.Length"), drop = 0.99, print = F)
predictAutoMAR  

Extract predictions and MAR columns from autoMAR objects

Description

this function can be used for autoMAR objects to generate the variable for missing variable not completely at random

Usage

predictAutoMAR(x, data, mar_var = NULL)

Arguments

x  
[autoMAR object | Required] autoMAR object for which prediction is desired

data  
[Data frame | Required] prediction data set to prepare the autoMAR outcomes

mar_var  
[character list | Optional] list of predefined mar variables

Value

flagged variables for missing not completely at random variable

Examples

## Missing at random features
train <- heart[1 : 199, ]
test <- heart[200 : 300, ]
marobj <- autoMAR (train, aucv = 0.9, strataname = NULL, stratasize = NULL, mar_method = "glm")

## print summary in console
testobj <- predictAutoMAR(marobj, test)

predictDataprep  

Extract predictions and generate columns from autoDataprep objects

Description

this function can be used for autoDataprep objects to generate the same for validation

Usage

predictDataprep(x, data)
Arguments

x [autoDataprep object | Required] autoDataprep object for which prediction is desired
data [data.frame | Required] prediction data set to prepare the MAR columns

Value

master data set same as train data set

Examples

```r
## Sample train data set
train <- heart[1:200, ]
test <- heart[201:350, ]
traindata <- autoDataprep(train, target = "target_var", missimpute = "default",
dummyvar = TRUE, aucv = 0.02, corr = 0.98, outlier_flag = TRUE,
interaction_var = TRUE, frequent_var = TRUE)
train <- traindata$master

## Predict same features for test set
test <- predictDataprep(traindata, test)
```

printautoDataprep

Print Method for the autoDataprep Class

Description

Print the result of autoDataprep object

Usage

printautoDataprep(x)

Arguments

x [Object | Required] an object of class autoDataprep

Value

Print summary autoDataprep function results on console
Examples

# Auto data prep
traindata <- autoDataprep(heart, target = "target_var", missimpute = "default",
dummyvar = TRUE, aucv = 0.02, corr = 0.98, outlier_flag = TRUE,
interaction_var = TRUE, frequent_var = TRUE)

# Print auto data prep object
printautoDataprep(traindata)

printautoMAR  
Print Method for the autoMAR Class  
Print the result of autoMAR object

Description

Print Method for the autoMAR Class Print the result of autoMAR object

Usage

printautoMAR(x)

Arguments

x  
[Object | Required] an object of class autoMAR

Value

Print summary of autoMAR output in console

Examples

## Missing at random features
marobj <- autoMAR(heart, aucv = 0.9, strataname = NULL, stratasize = NULL, mar_method = "glm")

## Print summary in console
printautoMAR(marobj)
smartEDA

Description

SmartEDA includes multiple custom functions to perform initial exploratory analysis on any input data describing the structure and the relationships present in the data. The generated output can be obtained in both summary and graphical form. The graphical form or charts can also be exported as reports.

Usage

```r
smartEDA(data, Template = NULL, Target = NULL, label = NULL,
          theme = "Default", op_file = NULL, op_dir = getwd(), sc = NULL,
          sn = NULL, Rc = NULL)
```

Arguments

data a data frame
Template R markdown template (.rmd file)
Target dependent variable. If there is no defined target variable then keep as it is NULL.
label target variable descriptions, not a mandatory field
theme customized ggplot theme (default SmartEDA theme) (for Some extra themes use Package: ggthemes)
op_file output file name (.html)
op_dir output path
sc sample number of plots for categorical variable. User can decide how many number of plots to depict in html report.
sn sample number of plots for numerical variable. User can decide how many number of plots to depict in html report.
Rc reference category of target variable. If Target is categorical then Pclass value is mandatory and which should not be NULL

Details

SmartEDA has four major functionalities 1. Descriptive statistics

- Numerical variable summary :
- ExpNumStat - Summary statistics for numerical variables `ExpNumStat`
- Categorical variable summary :
- ExpCatStat - Function provides summary statistics for all character or categorical columns in the dataframe `ExpCatStat`
• ExpCTable - Function to create frequency and custom tables ExpCTable

2. Data visualization
   • Numerical variable plot :
     • ExpNumViz - Distributions of numeric variables ExpNumViz
   • Categorical variable plot :
     • ExpCatViz - Distributions of categorical variables ExpCatViz
   • Normality testing plot:
     • ExpOutQQ - Quantile Quantile Plots ExpOutQQ
     • ExpParcoord - Parallel Coordinate plots ExpParcoord

3. Custom tables
   • Customized summary statistics :
     • ExpCustomStat - Customized summary statistics ExpCustomStat

4. EDA report
   • Function to create HTML EDA report :
     • ExpReport - Function to create HTML EDA report ExpReport

Value

HTML Rmarkdown output file in .html format

Source

Useful links:
   • CRAN page https://CRAN.R-project.org/package=SmartEDA
   • JOSS https://doi.org/10.21105/joss.01509

Examples

# Generate complete EDA report
smartEDA(iris, op_file="eda_report.html", op_dir = tempdir(), sc = NULL, sn = 2)
Index

*Topic datasets
  heart, 11
*Topic mlin
  heart.model, 12
  autoDataprep, 2
  autoMAR, 5
  autoMLmodel, 5
  autoMLReport, 8
  autoPDP, 8
  caret train, 7
  ExpCatStat, 17
  ExpCatViz, 18
  ExpCTable, 18
  ExpCustomStat, 18
  ExpNumStat, 17
  ExpNumViz, 18
  ExpOutQQ, 18
  ExpParcoord, 18
  ExpReport, 18
  FeatureEffects, 9
  generateFeature, 10
  heart, 11
  heart.model, 12
  impute, 4
  makeLearner, 7
  misspattern, 13
  mlr train, 7
  partial, 9
  plotPartialDependence, 9
  predictAutoMAR, 14
  predictDataprep, 14
  printautoDataprep, 15
  printautoMAR, 16
  smartEDA, 17
  tuneParams, 7