Package ‘KaggleHouse’

January 11, 2017

Type Package

Title Analysing House Prices in Ames, Iowa and Build a Sales Price Prediction Model

Version 0.1

Date 2016-11-05

Author@R c(person(``Frederik", ``Elischberger", role = c(``aut"),
email = ``frederik.elischberger@uni-muenster.de"),
person(``Marco", ``Niemann", role = c(``aut"),
email = ``marco.niemann@uni-muenster.de"))

Author Frederik Elischberger [aut],
Marco Niemann [aut]

Maintainer Marco Niemann <marco.niemann@uni-muenster.de>

Description Analysis functions for the Ames, Iowa dataset plus model building functions building on the analysis, used to create a model to predict house prices.

License What license is it under?

LazyData TRUE

Imports Rcpp (>= 0.12.7),
compiler,
ggplot2,
gridExtra,
dbscan,
mlr (>= 2.10),
methods,
magrittr,
Ckmeans.1d.dp,
mice (>= 2.25),
ParamHelpers,
glmnet,
irace,
FSelector,
h2o,
parallelMap,

Boruta,
Amelia,
xgboost (>= 0.6.2),
corrplot

Suggests kernlab,
randomForest

LinkingTo Rcpp

RoxygenNote 5.0.1

R topics documented:

`analysis.dotplot` *Generate Outlier Point Plot*

Description

This function generates an outlier point plot (for a description see slide 117 of the DA1 lecture of WT 2015/16) showing the value distribution of data `df`, alllowing to identify potential outliers.

Usage

```
analysis.dotplot(df, plot.title = "Outlier")
```

Arguments

`df`	`data.frame` with the data that should be analyzed for outliers.
`plot.title`	Title for the `ggplot` object created via this method.

Details

The method generates a dotplot as an univariate outlier detector for a given set of data `df` (must not be a `data.frame` or `matrix`. So via the function `analysis.dotplot` it computes the distribution of values in each given column via `table`. Based on the distribution the method `cpp_valueOccurrencesToPoints` will generate visualizable points representing each occurence in the form $(x_i, y_{i,j})$.

Value

ggplot point plot.

See Also

analysis.dotplot.all

cpp_valueOccurrencesToPoints

Examples

```
KaggleHouse:::analysis.dotplot(
   iris$Sepal.Length,
   plot.title = 'Outlier Plot for the Sepal Length of the Iris Dataset'
)
```

analysis.dotplot.all *Generate Outlier Point Plots*

Description

This function generates outlier point plots (for a description see slide 117 of the DA1 lecture of WT 2015/16).

Usage

```
analysis.dotplot.all(df, name = "dotplot.all", pdf = T)
```

Arguments

df	data.frame with the data that should be analyzed for outliers.
name	chr name for the file that can be optionally outputted via this function. The default name will be dotplot.all.
pdf	boolean indicating whether the functions results should be send to a PDF-file or whether they are intended for R- internal use. Default is output to PDF (option true).

Details

The method generates dotplots as univariate outlier detectors for all columns of a given data.frame df. So via the function analysis.dotplot it computes the distribution of values in each given column via table. Based on the distribution the method cpp_valueOccurrencesToPoints will generate visualizable points representing each occurence in the form $(x_i, y_{i,j})$.

Value

List of dotplots (ggplots) for pdf = F or a PDF file in the filesystem for pdf = T.

See Also

analysis.dotplot

cpp_valueOccurrencesToPoints

Examples

```
KaggleHouse:::analysis.dotplot.all(iris, name = 'iris.dots', pdf = T)
```

---

api.submit *Submit Results to Kaggle*

Description

Upload a preformatted csv file with the predicted test results to Kaggle.

Usage

```
api.submit(filepath, username, password, competition, message = "")
```

Arguments

`filepath`	Path to the csv file with the predicted data to be uploaded.
`username`	Kaggle username to login to the Kaggle plattform.
`password`	Kaggle password to login to the Kaggle plattform.
`competition`	Name of the Kaggle competition the submission is intended for.
`message`	Additional message that describes the submission. Is optional and by default empty.

Details

This method allows to upload the predicted output in csv format to the Kaggle competition. It bases on the Kaggle CLI tool by floydwch which is a Python CLI tool. This requires an existing Python installation as well as having the Kaggle CLI tool installed (with Python and pip via `pip install kaggle-cli`). With the requirements given this method works as an R wrapper around this tool.

See Also

Makes use of the Kaggle CLI tool by floydwch which can be found at https://github.com/floydwch/kaggle-cli.

Examples

```
KaggleHouse:::api.submit(
  filepath = "sample_submission.csv",
  username = "kaggle_user",
  password = "abc123",
  competition = "house-prices-advanced-regression-techniques"
)
```

cpp_bind *Combine two* NumericVector*s into a* NumericMatrix

Description

This function combines two given NumericVectors into a NumericMatrix. It thereby provides similar functionality as base::cbind.

Usage

```
cpp_bind(a, b)
```

Arguments

a	NumericVector representing the first column.
b	NumericVector representing the second column.

Value

NumericMatrix being the combination of the two NumericVectors a and b.

cpp_regex_selector_name *Extract Selector Name From Variable Name*

Description

This function extract the selector name (e.g. 'var') from a variable name like 'data$var'.

Usage

```
cpp_regex_selector_name(x)
```

Arguments

x	std::string containing the selector name (usually in the form (containingObject)$(selector)).

Details

This function basically takes variables names of the form `(containingObject)$(selector)` as an input. With a regex command the last selector - meaning the substring behind the last '$' - will be extracted and returned. In case that the input parameter `x` does not contain a selector, the empty string will be returned.

Value

`std::string` containing the extracted `selector` part. In case no selector can be found, the an empty string is returned.

Examples

```
# The outcome should be simply 'SalePrice'
KaggleHouse:::cpp_regex_selector_name(data_train_na$SalePrice)
```

`cpp_rep_na_chr` *Remove* `NA` *from* `CharacterVector`

Description

This function removes `NA` values from `CharacterVectors`.

Usage

```
cpp_rep_na_chr(xin, rep)
```

Arguments

`xin`	`CharacterVector` potentially containing removable `NA` values.
`rep`	`std::string` value to replace `NA`s occuring in `xin`.

Details

This function accepts a character input vector and then replaces each contained `NA` value with the `std::string` specified in the parameter `rep`.

Value

`CharacterVector` that is equal to `xin` except the `NA` values which have been replaced with `rep`.

Examples

```
KaggleHouse:::cpp_rep_na_chr(c("a", "b", NA), "c")
```

cpp_rep_na_num *Remove* NA *from* NumericVector

Description

This function removes NA values from NumericVectors.

Usage

```
cpp_rep_na_num(xin, rep)
```

Arguments

xin	NumericVector potentially containing removable NA values.
rep	int value to replace NAs occuring in xin.

Details

This function accepts a numeric input vector and then replaces each contained NA value with the int specified in the parameter rep.

Value

NumericVector that is equal to xin except the NA values which have been replaced with rep.

Examples

```
KaggleHouse:::cpp_rep_na_num(c(1, 2, NA), 3)
```

cpp_valueOccurrencesToPoints *Convert Value Occurences to Points*

Description

This function converts a data.frame of feature characteristic occurences to plottable points. For that the characteristic value is used as the x-value. To visualize multiple occurences, each of them is assigned a slightly higher y-position, so that multiple occurences of one characteristic will later on stack in the plotted graph.

Usage

```
cpp_valueOccurrencesToPoints(x)
```

Arguments

x	data.frame with the results of the table command on a given vector. x should contain the counts per characteristic of a specific feature.

Value

List including plottable point positions representing the counts of feature characteristics.

Examples

```
val <- rep(1:5, c(10, 20, 30, 40, 50))
df <- as.data.frame(table(val))
KaggleHouse:::cpp_valueOccurrencesToPoints(df)
```

create_barplot *Create Bar Plot*

Description

This method acts as a wrapper around the ggplot2::ggplot.

Usage

```
create_barplot(data, col_name)
```

Arguments

data	vector with the data of the column col_name.
col_name	character string with the name of the column/feature for which the bar plot is created.

Details

This method creates a ggplot2::ggplot for the provided data. The plotting function is preconfigured to create a bar plot with ggplot2::theme_light and vertically aligned characteristics on the x-axis (so that all can be displayed).

Value

ggplot2 bar plot for the feature col.name and data data.

Examples

```
KaggleHouse:::create_barplot(data_train$MSZoning, "MS Zoning")
```

data.generate_data_views *Data view generation function*

Description

This function creates five different variables containing the names of features.

Usage

```
data.generate_data_views()
```

Details

Calling this function is optional. It is useful when analysing different types of features as e.g. continuous, discrete, nominal, ordinal and the target feature Saleprice. All variables are bound the base environment.

echo *Set Cat Function to Echo variable*

Description

This function assigns the alias 'echo' to the cat-function.

Usage

```
echo(..., file = "", sep = " ", fill = FALSE, labels = NULL,
  append = FALSE)
```

echoln *Write String to Output with appended Newline*

Description

This function can be used to write a string `str` to the output with an automatically appended newline.

Usage

```
echoln(str)
```

Arguments

`str`	with the content to print to the output

`feature.boruta` *Boruta Feature Selection - Wrapper*

Description

Convenience method that calls `feature.boruta.comp` with preset parameters that are commonly used for learners in this package.

Usage

```
feature.boruta(data = data_train_numeric_clean_imputed, recompute = F,
  desc = "")
```

Arguments

`data`	`data.frame` containing the data on which the Boruta feature selection should be executed. Will be used to feed the `target` and `predictors` variables of `feature.boruta.comp`. Defaults to `data_train_numeric_clean_imputed`.
`recompute`	`boolean` switch that determines if the Boruta feature selection should be repeated when the results of a prior run have been found. Recommended when parameters have changed. Defaults to `FALSE`.
`desc`	Additional comment that can be appended to the name of the saved Boruta object. Can e.g. be used to store different Boruta runs for different learners. Defaults to the empty string.

Details

This method executes the packages Boruta wrapper `feature.boruta.comp` with all parameters preset to fit the needs of the package learners. Furthermore it stores each Boruta object after the computation to the `output/feature_selection` directory. Before a new computation is started the directory is checked for the existance of an already computed Boruta object. If one is available and the `recompute` flag is `FALSE` the previously computed object is loaded and used. Finally the method binds all selected features (confirmed and tentative ones) to the `.GlobalEnv` as the `features_boruta` variable.

Examples

```
KaggleHouse:::feature_boruta()
KaggleHouse:::feature_boruta(recompute = T)
KaggleHouse:::feature_boruta(recompute = T, desc = "_test_run_")
```

feature.boruta.checkInputParams *Check Boruta Feature Selection Input Parameters*

Description

The method ensures that the parameters of the feature.boruta function are valid.

Usage

```
feature.boruta.checkInputParams(target, predictors, checkNA = F,
  verbose = F)
```

Arguments

target	Response vector; factor for classification, numeric vector for regression.
predictors	data.frame with predictors.
checkNA	boolean switch that decides whether the method will conduct a check for NA values or not. Default is FALSE.
verbose	boolean switch that decides whether the error output will provide more verbose information. Default is FALSE.

Details

This method analyses the target and predictors input variables of the feature.boruta function. It conducts two mandatory checks and one optional check. The first of the two mandatory checks evaluates whether the correct data types are used. So the target variable should be a vector while the predictors variable is supposed to be a data.frame. As a second check it is ensured that both variables have the same number of rows, which implies that the number of considered observation is equal for both. If one of the checks fail, execution will be aborted with an error explaining the reasons. The optional check evaluates the presence of NA values as these are not useable for many of the Boruta classifiers. In case either target and/or predictors contain only one NA a warning is issued. If the verbose switch is TRUE, the error will provide a list of predictors columns which contain NA values.

Examples

```
KaggleHouse:::feature.boruta.checkInputParams(
  target = data_train_na$SalePrice, predictors = data_train_na[-81],
  checkNA = T, verbose = T
)
```

feature.boruta.comp *Boruta Feature Selection*

Description

Wrapper around the Boruta package. Boruta is a so called all relevant feature seletion wrapper, capable of working with each classifier outputting variable importance measure (VIM). This function provides a wrapper ensuring correct provision of input data and the potential to execute convenience functions that e.g. provide regression formula output.

Usage

```
feature.boruta.comp(target, predictors, fixNA = F, roughFix = F,
  variables = F, selected = F, formula = F, tentative = F,
  pValue = 0.01, mcAdj = T, maxRuns = 100, doTrace = 0,
  holdHistory = T, getImp = Boruta::getImpRfZ, verbose = F, ...)
```

Arguments

`target` Response vector; factor for classification, numeric vector for regression.

`predictors` `data.frame` with predictors.

`fixNA` boolean switch that decides how `NA` values in the `predictors` and `target` variables will be handled. `FALSE` would cause the `NA`s to be ignored. `TRUE` will eliminate all observations including a `NA` value. Default is `FALSE`.

`roughFix` boolean switch that decides whether the `Boruta::TentativeRoughFix` method will be used to resolve potentially remaining undecided variables. Default is `FALSE`.

`variables` boolean switch that decides whether the variables of all three categories (Confirmed, Tentative, Rejected) will be appended to the returned `Boruta` object. Default is `FALSE`.

`selected` boolean switch that decides whether the confirmed and tentative varialbes will be added as a combined vector to the `Boruta` object. Default is `FALSE`. Only works when `variables` is `TRUE`.

`formula` boolean switch that decides whether a formula will be appended to the returned `Boruta` object. This formula will relate the `target` with all confirmed `predictors`. Depending on the `tentative` switch the tentative variables might be added as well. Defaults to `FALSE`.

`tentative` boolean switch that decides whether tentative attributes will be considered for a formula. Default is `FALSE`.

`pValue` Confidence level. Default value should be used. Default is 0.01.

`mcAdj` If set to `TRUE`, a multiple comparisons adjustment using the Bonferroni method will be applied. Default value should be used; older (1.x and 2.x) versions of Boruta were effectively using FALSE. Default value is `TRUE`.

maxRuns Maximal number of importance source runs. You may increase it to resolve attributes left tentative. Default is 100.

doTrace Verbosity level. 0 means no tracing, 1 means reporting decision about each attribute as soon as it is justified, 2 means same as 1, plus reporting each importance source run. Default is 0.

holdHistory If set to TRUE, the full history of importance is stored and returned as the ImpHistory element of the result. Can be used to decrease a memory footprint of Boruta in case this side data is not used, especially when the number of attributes is huge; yet it disables plotting of such made Boruta objects and the use of the Boruta::TentativeRoughFix function. Default is FALSE.

getImp Function used to obtain attribute importance. The default is getImpRfZ, which runs random forest from the ranger package and gathers Z-scores of mean decrease accuracy measure. It should return a numeric vector of a size identical to the number of columns of its first argument, containing importance measure of respective attributes. Any order-preserving transformation of this measure will yield the same result. It is assumed that more important attributes get higher importance. +-Inf are accepted, NaNs and NAs are treated as 0s, with a warning. Default is Boruta::getImpRfZ.

verbose boolean switch that decides whether the error output will provide more verbose information. Default is FALSE.

Details

The method first saves the name of the original target parameter so it is potentially reusable for formula creation later on. In case the fixNA switch is TRUE, all observations containing NA values will be eliminated. If this should affect all observations an error will be produced. Before executing the Boruta algorithm, the important input parameters target and predictors will be checked via the feature.boruta.fixNA method. Should any issues with the input be found (wrong data types, differing lengths, NAs) an appropriate error will be thrown. Next the actual Boruta::Boruta algorithm is executed with the provided parameters. Bortua than iteratively compares the importance of shadow attributes with the original attributes. Those with a significantly worse performance than shadow attributes will be rejected; those performing significantly better will be confirmed. Since the Boruta algorithm might not converge in the given maxRuns iterations, the Boruta::TentativeRoughFix can be used to resolve still missing values (given roughFix is TRUE). Finally, depending on the values of the variables and formula switches, a formula will be created and/or the confirmed/rejected/tentative attributes are appended to the returned Boruta object.

Value

Boruta object as it is also returned by the underlying Boruta::Boruta method. This default return value can include severeal extensions, depending on parameters like formula:

target The name of the target vector.

variables Variable names of all three categories (Confirmed, Tentative, Rejected)

selected Variables names of confirmed and tentative variables in one vector.

formula Formula of the form target ~ predictors.(confirmed/tentative)

References

Miron B. Kursa, Witold R. Rudnicki (2010). Feature Selection with the Boruta Package. *Journal of Statistical Software, 36(11)*, p. 1-13. URL: `http://www.jstatsoft.org/v36/i11/`

See Also

`Boruta::Boruta`

`feature.boruta.fixNA`

`feature.boruta.tentative`

`feature.boruta.variables`

`feature.boruta.selected`

`feature.boruta.formula`

`feature.boruta.checkInputParams`

Examples

```
KaggleHouse:::feature.boruta(
  target = data_train_na$SalePrice, predictors = data_train_na[-81],
   fixNA = T, roughFix = T, verbose = T
)
```

`feature.boruta.fixNA` *Remove* `NA` *Containing Observations*

Description

Filter all observations from the `target` and `predictors` variable that contain `NA` values to obtain a `NA` dataset usable by the Boruta classifiers.

Usage

```
feature.boruta.fixNA(target, predictors)
```

Arguments

`target`	Response vector; factor for classification, numeric vector for regression.
`predictors`	`data.frame` with predictors.

Details

The method first carries out a reduced parameter check via `feature.boruta.checkInputParams`. Afterwards both `target` and `predictors` are combined to a unified `data.frame`. This `data.frame` is then cleaned from `NA`s by the `na.omit` function. Before splitting the `data.frame` into a target and a predictor variable again, it is checked that at least one observation prevailed. If that is not the case, the method is aborted with an error.

Value

The NA-cleaned data.frame will be returned with two additional parameters:

target	Containing the NA-cleaned target variable.
predictors	Containing the NA-cleaned predictors.

Examples

```
 KaggleHouse:::feature.boruta.fixNA(
   target = data_train_na$SalePrice, predictors = data_train_na[-81]
)
```

feature.boruta.formula

Create Formula Based on Boruta Selected Features

Description

The method creates a formula that puts a target value in relationship to features selected by the Boruta algorithm.

Usage

```
feature.boruta.formula(boruta, tentative = F)
```

Arguments

boruta	Boruta object obtained by the execution of feature.boruta method.
tentative	boolean switch to decide whether tentative variables should be part of the created formula. Default is FALSE.

Details

This method evaluates the Boruta object. In case some other object is provided to the function it will abort with an error. Given correct inputs the function accesses the target property of the custom Boruta object returned by feature.boruta and also obtains the categorized list of the evaluated variables via feature.boruta.variables. With these inputs a formula of the structure target ~ predictorVar1 + ... is created which can then e.g. be reused for regression.

Value

formula object that represents the relationship between the predictors and the target variable which has been evaluated by the Boruta algorithm. The form will be target ~ predictorVars where predictorVars will only be confirmed (and tentative) variables.

Examples

```
boruta <- KaggleHouse:::feature.boruta(
  target = data_train_na$SalePrice, predictors = data_train_na[-81],
   fixNA = T, roughFix = T, verbose = T
)

KaggleHouse:::feature.boruta.formula(boruta, tentative = T)
```

`feature.boruta.report` *Create PDF and Text Reports About Selected Boruta Features*

Description

This methods provides a PDF with the major info graphics related to Boruta feature selection, as well as a text file with all confirmed, tentative and rejected variables including the formula generated from them.

Usage

```
feature.boruta.report(boruta)
```

Arguments

`boruta`	The `Boruta` object for which a report should be generated.

Details

In a first step this method will generate two plots for the given `Boruta` object: One plot that will show boxplots for all features, as well as their importance and their selection status (e.g. a red feature has been discared, a green one confirmed). Additionally the imputation history is added, which shows the importance (and acceptance status) of each feature over the time. These two graphics are combined into one PDF file. Since it is versioned with time information, each call to this method will automatically generate a new report. The time stamp thus allows to version different `Boruta` objects. In case the `Boruta` object has been generated with `feature.boruta.comp` and contains the `Confirmed` (etc.) variables an additional text file is generated and filled with all the information regarding those features. If the `formula` switch has been activated the generated formula will also be added to that text file. Similar to the PDF file the txt file is versioned via timestamps as part of the file name.

Examples

```
boruta <- KaggleHouse:::feature.boruta(
  target = data_train_na$SalePrice, predictors = data_train_na[-81],
   fixNA = T, roughFix = F, variables = T, selected = T, formula = T,
   verbose = T
)
KaggleHouse:::feature.boruta.report(boruta)
```

feature.boruta.selected *Provide all Non-Rejected Boruta Features*

Description

This method combines the features that have either been confirmed or have been marked as tentative in the Boruta algorithm into one variable.

Usage

```
feature.boruta.selected(boruta)
```

Arguments

boruta	The Boruta object to which the Selected variable should be attached.

Details

This method usually is not meant to be applied by a package user. In case it is still used to obtain the confirmed and tentative variables of a Boruta object it has to be ensured that it has the variables Confirmed and Tentative. These can be obtained via the variables switch of the feature.boruta.comp method that has to be TRUE. Then the features contained in both variables will be combined into a Selected variable. Finally the method binds the Selected variable to the .GlobalEnv as features_boruta.

Value

Boruta object with an attached Selected variable containing the confirmed and tentative features.

Examples

```
boruta <- KaggleHouse:::feature.boruta(
  target = data_train_na$SalePrice, predictors = data_train_na[-81],
   fixNA = T, roughFix = F, variables = T, verbose = T
)
KaggleHouse:::feature.boruta.selected(boruta)
```

feature.boruta.tentative

Check For Tentative Variables

Description

The method checks a Boruta object for the decision status by checking if some variables are still tentative.

Usage

```
feature.boruta.tentative(boruta)
```

Arguments

boruta	Boruta object obtained by the execution of feature.boruta method.

Details

This method evaluates the Boruta object. In case some other object is provided to the function it will abort with an error. All correctly assigned objects will then be checked for variables that still have the tentative status. If at least one is still tentative a TRUE will be returned.

Value

boolean value that is TRUE as soon as one tentative variable is found. FALSE otherwise.

Examples

```
boruta <- KaggleHouse:::feature.boruta(
  target = data_train_na$SalePrice, predictors = data_train_na[-81],
   fixNA = T, roughFix = T, verbose = T
)

KaggleHouse:::feature.boruta.tentative(boruta)
```

feature.boruta.variables

Obtain List of Confirmed, Tentative and Rejected Variables

Description

The method obtains a list of three vectors containing confirmed, tentative and rejected variables.

Usage

```
feature.boruta.variables(boruta)
```

Arguments

boruta	Boruta object obtained by the execution of feature.boruta.comp method.

Details

This method evaluates the Boruta object. In case some other object is provided to the function it will abort with an error. From correctly provided Boruta objects a list of all evaluated variable names will be obtained. Based on the finalDecision variable of the Boruta object the variable names will be split into three vectors representing confirmed, tentative and rejected variables. These vectors are combined in a list and are returned as such.

Value

A list with the three entries:

Confirmed	Vector of confirmed variables.
Tentative	Vector of tentative variables.
Rejected	Vector of rejected variables.

Examples

```
boruta <- KaggleHouse:::feature.boruta(
  target = data_train_na$SalePrice, predictors = data_train_na[-81],
   fixNA = T, roughFix = T, verbose = T
)

KaggleHouse:::feature.boruta.variables(boruta)
```

feature.lasso *Lasso feature selection*

Description

When executed an optimal set of features according to the lambda value that minimizes the loss is bound to a global variable features_lasso.

Usage

```
feature.lasso(data = data_train_numeric_clean_imputed)
```

Arguments

data	Input data

Details

Default dataset is data_train_numeric_clean_imputed. The family is gaussian and for measuring the goodness auc is chosen.

Examples

```
feature.lasso(data=BostonHousing)
```

---

general_barplot *Generate Bar Plots for Dataset Analysis*

Description

This method generates bar plots for the nominal and ordinal variables of a given dataset to visualize their value distributions.

Usage

```
general_barplot(data = data_train, data_cols = c(iowa.houses.nominal.vars,
  iowa.houses.ordinal.vars), name = "barplots", pdf = T)
```

Arguments

data	data.frame containing the data for which the bar plots should be created.
data_cols	vector of column names containing nominal and ordinal data (for which the bar plots should be generated).
name	character string that is used as the filename.
pdf	boolean switch to either export the graphics to a PDF file or print them on screen. Defaults to TRUE.

Details

This method is used to visualize the distribution of nominal and ordinal features in a given dataset data. As not automated detection of these features is conducted, the columns containing such data are given by the data_cols parameter. For each of the features a bar plot is created via the create_barplot method. If the pdf switch is activated (switch == TRUE) the bar plots will be exported to a PDF file. Here the filename is determined by the name attribute. Each PDF page will then contain up to two bar plots. Otherwise the bar plots will be shown on the screen.

Examples

```
# Create vectors containing the column names of columns with oridinal or
# nominal features.
KaggleHouse:::data.generate_data_views()
# Generate PDF bar plots.
KaggleHouse:::general_barplot()
```

general_hist *Create Histogram of Numeric Data (to visualize Distribution)*

Description

The function creates histograms for all columns/variables with numeric variables in the given dataset `x` (at least this is the default behaviour) and saves them to a PDF file. Creating the histograms helps to get a better understanding of the distribution of the values. As the visualized distributions might ressemble the normal distribution, the p-value is plotted below the histograms as an additional indicator for normality.

Usage

```
general_hist(x, name, pdf = T)
```

Arguments

`x`	dataset to be analysed

general_plot *Create Scatterplot of Data (to visualize Correlations)*

Description

The function creates a scatterplot of the complete dataset given by the variable and saves it in a PDF file. The lower panel makes use of `panel.smooth` to create the usual scatterplots for pairs of two variables but also adds some kind of a 'regression line' to the plot to increase the understandability. The upper panel uses a custom panel that shows of the numeric correlation values (highlighted with different font sizes and colors).

Usage

```
general_plot(x, name, threshold = 0.7, pdf = T, o = T,
  cor.met = "pearson")
```

Arguments

`x`	Dataset to be analysed.
`name`	Name that will be used in case a PDF will be produced.
`threshold`	The minimum correlation that two features must have to be considered further. If a feature does not have a correlation greater than the threshold with at least two other features it will be excluded from the correlation plot. Defaults to a value of 0.7.
`pdf`	Switch variable that decides whether the plot will be output to PDF or not. Defaults to PDF output.

o	Boolean switch to decide whether the method is allowed to print progress output to stdout. Defaults to true.
cor.met	Methodology used to compute the correlation. Can be "pearson", "spearman" oder "kendall". Defaults to "pearson".

general_qq *Create a QQ-Plot of Numeric Data (to Analyse the Normality Assumption)*

Description

This function takes a dataset x - which by default will consist of all numeric variables of the Tripadvisor hotel dataset - and creates a QQ-plot for each of the contained columns/variables. To improve the ease of understanding whether the given data is rather normal distributed or not (theoretical and practical quantiles are similar) additional QQ-Lines are added.

Usage

```
general_qq(x, name, pdf = T)
```

Arguments

x	dataset to be analysed

general_summary_detail *Data Summary - Get a Summary of the Data*

Description

This function executes the summary-function on inputted datasets x. The results are written to a csv-file for later use.

Usage

```
general_summary_detail(x, name)
```

Arguments

x	dataset to be analysed

imputation.test *Impute tst data*

Description

Imputation function for the test dataset using package mice and manual substitution to remove missing values.

Usage

```
imputation.test(data = data_test_numeric_clean)
```

Arguments

data	Input data which is set by default to data_test_numeric_clean

Details

Since some features are not missing at random imputation is not an appropriate approach. Therefore the features MasVnrArea, LotFrontage, Electrical and GarafeYrBlt are manually imputed. The remaining features that still contain at least one missing value are solely imputed using mice with 50 maximum iterations with seed 500 using predictive-mean matching. After the execution one can access the variable data_train_numeric_clean_imputed.

imputation.train *Impute training data*

Description

Imputation function for the training dataset using package mice and manual substitution to remove missing values.

Usage

```
imputation.train(data = data_train_numeric_clean)
```

Arguments

data	Input data which is set by default to data_train_numeric_clean

Details

Since some features are not missing at random imputation is not an appropriate approach. Therefore the features MasVnrArea, LotFrontage, Electrical and GarafeYrBlt are manually imputed. The remaining features that still contain at least one missing value are solely imputed using mice with 50 maximum iterations with seed 500 using predictive-mean matching. After the execution one can access the variable data_train_numeric_clean_imputed.

`KaggleHouse` *Kaggle House Analytial Package*

Description

Analyse

Details

Analyse.

`learner.deeplearning` *Deep Learning with h2o tuned by mlr*

Description

This method performs parameter tuning and feature selection on the provided `data` data set.

Usage

```
learner.deeplearning(data = data_train_numeric_clean_imputed)
```

Arguments

`data`	`data.frame` containing the data that should be used for deep learning. For optimal results an imputed and cleaned data set should be provided. The default data set is the KaggleHouse train data set after imputation and conversion into a numeric `data.frame`.

Details

This method trains an ANN based on the `h2o` package (more specificially the `h2o.deeplearning`) function. For the training it uses the given `data` and tries to adjust the `hidden` (number of hidden layers) and the `rate` (learning rate) parameters. On top feature selection will be performed to only keep those features actually contributing to the model. The final results will be saved to the `learner.deeplearning_result.RData` file. This way they can later be reused to extract the optimal parameters for a deeplearning ANN.

References

A. Candel, J. Lanford, E. LeDell, V. Parmar, A. Arora (2015). Deep Learning with H2O (*Third Edit.*) Publisher: *H2O.ai, Inc.* URL: `http://h2o.gitbooks.io/deep-learning/`

S. Aiello, T. Kraljevic and P. Maj (2016). h2o: R Interface for H2O URL: `http://www.h2o.ai/`

Examples

```
KaggleHouse:::learner.deeplearning(data_train_numeric_clean_imputed)
```

`learner.lasso` *Basic glmnet learner*

Description

Parallel tuning function using glmnet that is either based on glmnet, boruta or the whole dataset.

Usage

```
learner.lasso(data = data_train_numeric_clean_imputed, lasso = FALSE,
  boruta = FALSE)
```

Arguments

`data`	Input data which is default set to the numeric, imputed and cleaned training dataset
`lasso`	Boolean flag that shrinks data to the features of `feature.lasso()` stored in the variable `features_lasso`
`boruta`	Boolean flag that shrinks data to the features of `boruta.lasso()` stored in the variable `features_boruta`

Details

The default execution uses the whole training dataset. By setting either the `lasso` or `boruta` parameter to true the number of features is reduced according to the results of the feature selection. The glmnet learner is wrapper in a Filter wrapper that uses chi squared as feature selection method. The result is three times cross validated at maximum 2000 experiments using irace as a control structure.

Value

0 as error output if both flags are set to true

Examples

```
KaggleHouse:::learner.lasso(lasso=TRUE)
```

learner.stacked *Stacked learner*

Description

Combines the parameter sets from the tuning of the basic learners to predict the final Saleprice using linear regression.

Usage

```
learner.stacked(input_train = data_train_numeric_clean_imputed,
  input_test = data_test_numeric_clean_imputed)
```

Arguments

`input_train` Input data which is by default set to `data_train_numeric_clean_imputed`

`input_test` Input data which is by default set to `data_test_numeric_clean_imputed`

Details

Uses the `features_boruta` to select only features that are considered important by `feature.boruta()`. This stacked learner uses xgboost, lasso with the tuned parameters and deeplearning with 10 hidden layers each containing 300 nodes as basic learners. At last linear regression is used to predict the Saleprice using all three predictions. The result is stored in `final_submission_stacked_learner.csv` and can be directly uploaded to kaggle.

Examples

```
KaggleHouse:::learner.stacked()
```

learner.xgboost *Basic xgboost learner*

Description

Parallel tuning function using xgboost that is either based on glmnet, boruta or the whole dataset.

Usage

```
learner.xgboost(data = data_train_numeric_clean_imputed, lasso = FALSE,
  boruta = FALSE)
```

Arguments

`data` Input data which is default set to the numeric, imputed and cleaned training dataset

`lasso` Boolean flag that shrinks data to the features of `feature.lasso()` stored in the variable `features_lasso`

`boruta` Boolean flag that shrinks data to the features of `boruta.lasso()` stored in the variable `features_boruta`

Details

The default execution uses the whole training dataset. By setting either the `lasso` or `boruta` parameter to true the number of features is reduced according to the results of the feature selection. The number of rounds is set to 500 because including that parameter into the set of tuning parameters leads to worse results. The xgboost learner is wrapper in a Filter wrapper that uses chi squared as feature selection method. The result is three times cross validated at maximum 1000 experiments using irace as a control structure.

Value

0 as error output if both flags are set to true

Examples

```
KaggleHouse:::learner.xgboost(lasso=TRUE)
```

`plot_against_var` *2D plot generator*

Description

Generates a set of two dimensional plots where one variable is plotted against all other variables.

Usage

```
plot_against_var(df, var, pdf = TRUE)
```

Arguments

`df` Dataframe containing all features to which `var should be plotted`

`var` Vector which is the variable that is plotted to each variable in `df`

`pdf` Boolean variable stating whether to create a pdf or not

Details

All two dimensional plots are saved in one pdf file named `understand_data.data_against` using ggplot. This function is typically used to find correlations to the target variable.

prepare.transform_data

Data transformation function

Description

Substitutes missing values of a specific set of columns with the character "None".

Usage

```
prepare.transform_data(data, na.col)
```

Arguments

`data`	Input data which contains missing values
`na.col`	Set of column names which contain at least one missing value that is then substituted with the character "None"

Details

Especially in the given dataset missing values often yielded semantical meaning. To generate an additional class when transformed into numerical values this function is used to substitutes those missing values not at random with a character "None" that should get transformed into an numeric value instead of NA.

Examples

```
KaggleHouse:::prepare.transform_data(data.frame(id = c(1,2,NA)), "id")
```

preprocess.generate_cleaned_data

Cleaning data function

Description

Main purpose is to remove a set of rows and columns from a given dataset.

Usage

```
preprocess.generate_cleaned_data(data, rows, cols)
```

Arguments

`data`	Input data that is supposed to be reduced in number of columns and rows
`rows`	Rows that should be removed from `data`
`cols`	Columns that should be removed from `data`

Details

After checking for valid input parameter first rows and then columns are removed from the `data`. The cleaned dataset is globally available under the name of the input dataset with "_clean" as prefix.

Value

`data` with removed rows and columns

`rda.conversion.checkRawData` *Get List of RAW data files.*

Description

The function checks the given `directory` for RAW data files.

Usage

```
rda.conversion.checkRawData(directory = "inst/extdata")
```

Arguments

`directory`	The directory in which the RAW files should be searched.

Details

First the presented `directory` is checked for existance. In case it cannot be found a `warning` will be thrown. If it is present, a list of all files in the directory will be created and returned. These files will then be considered as candidates for being loaded as package data.

Value

`List` of files present in the specified `directory`.

`rda.conversion.convertData` *Convert RAW data to RDA data.*

Description

This function converts RAW data into the package-typical RDA format.

Usage

```
rda.conversion.convertData(directory = "inst/extdata")
```

Arguments

directory	The directory in which the RAW files should be searched.

Details

This function is a wrapper around all other functions with the `rda.conversion` prefix. It starts by calling `rda.conversion.ensureDataDir` to ensure that the target directory for the RDA files exists. Once that is assured the specified `directory` will be checked via `rda.conversion.checkRawData` for its existance. In case it exists the list of contained files will be returned and be iteratively loaded via `rda.conversion.loadDataFile` and saved as a RDA via `rda.conversion.saveAsRDA`.

See Also

`rda.conversion.ensureDataDir`

`rda.conversion.checkRawData`

`rda.conversion.loadDataFile`

`rda.conversion.saveAsRDA`

---

rda.conversion.ensureDataDir

Ensure Existance of Data Directory

Description

The function checks the existance of the `data` directory and creates it in case it is necessary.

Usage

```
rda.conversion.ensureDataDir()
```

Details

The function ensures the presence of a `data` directory via the utility helper function `util.ensure.dir`. This method then checks the availability of the folder and creates a new folder in case it should be necessary.

See Also

`util.ensure.dir`

rda.conversion.loadDataFile

Load data from RAW data file

Description

A function that loads raw data from a CSV file to a local variable.

Usage

```
rda.conversion.loadDataFile(directory, file, header = T, sep = ",",
  dec = ".", quote = "\"")
```

Arguments

`directory` The directory in which the target `file` is located.

`file` The file that should be loaded by this function.

`header` `boolean` specifying whether the to-read-CSV has a header row or not. Defaults to `TRUE`.

`sep` Character specifying the `chr` separating two data entries. Defaults to `,`.

`dec` Character specifying the decimal split `chr`. Defaults to `.`.

`quote` Set of quoting characters. Defaults to `""`.

Details

The function first checks the existance of the given `directory` and `file`. In case both exist, the specified `file` will be loaded considering the loading parameters `header, sep, dec` and `quote`.

Value

`data.frame` containing the data present in the specified `file`.

Examples

```
# Assuming the existance of directory 'dir' and file 'data.csv':
KaggleHouse:::rda.conversion.loadFile(
    'dir', 'data.csv', header = T, sep = ";", dec = ".", quote = '\"'
)
```

rda.conversion.saveAsRDA *Convert Loaded Data to RDA File.*

Description

This function converts a loaded dataset dat into an RDA file.

Usage

```
rda.conversion.saveAsRDA(dat, name)
```

Arguments

dat	data.frame containing the data loaded from a raw data/CSV- file.
name	Name assigned to dat by the filename of it's raw file.

Details

This function takes a dataset dat (typically previously loaded via rda.conversion.loadDataFile) and the name it comes with. Based on this it first checks the data directory for an RDA file with the same name (indicating that the data in question has already been converted). Whenever conversion is still required, the input data dat will be bound to the .GlobalEnv under the label of name. This allows to store the variable content to an RDA file via base::save. Finally .GlobalEnv will be cleaned from dat again.

See Also

rda.conversion.loadDataFile

Examples

```
# Assuming the existance of directory 'inst/extdata' and file 'data.csv':
imp <- KaggleHouse:::rda.conversion.loadDataFile('inst/extdata', 'data.csv')
KaggleHouse:::rda.saveAsRDA(imp, 'data.csv')
```

run_descriptive *Descriptive analysis*

Description

Mainly used to output plots that are used to get a general understanding of the given data.

Usage

```
run_descriptive(train = data_train,
  train_no_na = data_train_numeric_clean_imputed)
```

Arguments

`train`	Training dataset
`train_no_na`	Training dataset only containing numerical values and no missing data

Details

The plots and text documents are saved under the /output directory. Besides generating a textual summary of the whole training dataset this function also generates histograms, barplots, scatterplots and qq-plots.

`run_generate_data` *Function for generating datasets*

Description

In general the first function that should be executed when the package is loaded. Generates the two dataset `data_train_numeric_clean_imputed` and `data_test_numeric_clean_imputed`.

Usage

```
run_generate_data()
```

Details

First the function generates data views for e.g. distinguishing categorical and ordinal variables. Next the data is cleaned by removing unnecessary columns and rows. After that imputation takes place by manually and automatically(mice with pmm) substituting missing values.

`run_learner` *Main stacked learner*

Description

Executes the stacked learner on the train and test dataset

Usage

```
run_learner(train = data_train_numeric_clean_imputed,
  test = data_test_numeric_clean)
```

Arguments

`train`	Cleaned, imputed and numeric training dataset
`test`	Cleaned, imputed and numeric test dataset

Details

Wrapper function that is exported to the user of the package.

util.contain_na *Check Data Columns for NA Values*

Description

This method returns a logical vector that indicates the presence of NA values in every column of data.

Usage

```
util.contain_na(data)
```

Arguments

data	data.frame in which each column should be checked for the presence of NA values.

Details

This method takes a data.frame as an input and then checks for each column if it contains any NA values. The results get stored in a logical vector (always together with the column name).

Value

Named logi (being a logical vector also containing the column name for each logical value) indicating the presence of NA values for each column of data.

Examples

```
KaggleHouse:::util.contain_na(data_train)
```

util.ensure.dir *Ensure Existance of Directory*

Description

This method ensures that the directory given as a parameter exists.

Usage

```
util.ensure.dir(dir.name)
```

Arguments

dir.name	character string of the name of the directory which should be checked for existance and added if missing.

Details

This method takes a `character` string as input which denotes a directory path. It first checks whether the given directory already exists on the filesystem. If this is not the case it will be created.

Examples

```
KaggleHouse:::util.ensure.dir("output")
```

`util.generate.submit` *Generate a Kaggle-submitable CSV from the Predicted Data*

Description

This method generates a CSV file submitable to Kaggle from a `data.frame` containing the relevant data (incl. the predictions).

Usage

```
util.generate.submit(data, name)
```

Arguments

`data`	`data.frame` containing the data needed for a submission of predictions to the Kaggle platform.
`name`	`character` string that represents the name of the CSV file to which the submission data will be stored.

Details

This method takes a `data.frame` that already contains all relevant data for a prediction submission to the Kaggle platform. E.g. in case of the "House Prices: Advanced Regression Techniques" challenge this `data.frame` would have to include one column with the ID and one with the predicted sale price of a house. These data is then written to a CSV file via a preconfigured call to the `write.csv` function to a file with the name given in the `name` parameter.

`util.list_to_df` *Convert List of Matrices to* `data.frame`

Description

Transforms a list of matrices (with the same columns ['objects', 'counts']) into a concatenated `data.frame`.

Usage

```
util.list_to_df(lst)
```

Arguments

`lst`	List with matrices.

Details

This method first uses the `util.list_to_matrix` method to convert the given list of matrices to the concatenated version of all matrices stored in a single matrix. Afterwards the derived matrix is transformed into the `data.frame` returned by the function.

Value

`data.frame` being the concatenated version of all matrices in `list` `lst`.

`util.list_to_matrix` *Convert list of matrices to matrix*

Description

Transforms a list of matrices (with the same columns ['objects', 'counts']) into a concatenated matrix.

Usage

```
util.list_to_matrix(lst)
```

Arguments

`lst`	`list` with matrices.

Value

`matrix` being the concatenated version of all matrices in `list` `lst`.

util.mse *Compute the MSE between two `data.frame`s*

Description

This method is used to compute the mean squared error (MSE) between two `data.frames`.

Usage

```
util.mse(data_predicted, data_original)
```

Arguments

`data_predicted`	First `data.frame` for the MSE computation.
`data_original`	Second `data.frame` for the MSE computation.

util.na.rm *Clear Matrix of all `NA` rows.*

Description

Function to remove rows from a given matrix `m` which contain a specified amount of `NA` values. The default is that a row is removed from the matrix as soon as one `NA` value is included. But it is also possible to configure the method to remove rows only when all values are `NA`.

Usage

```
util.na.rm(x, fun = any)
```

Arguments

`x`	with the matrix to clear from `NA` values
`fun`	with the function to determine the number of allowed `NA`s

Value

Matrix with cleaned rows

util.number_na *Count Number of* NA *Values in a* data.frame

Description

This method counts the number of NA values for each column of a given data.frame.

Usage

```
util.number_na(data)
```

Arguments

data	data.frame in which the number of NA values is counted for each column.

Details

This method takes a data.frame as an input and then counts the number of NA values in each column. The results get stored in a numeric vector (always together with the column name).

Value

Named num (being a numeric vector also containing the column name for each numeric value) indicating the number of NA values for each column in data.

Examples

```
KaggleHouse:::util.number_na(data_train)
```

util.panel.cor *Define a Scatterplot Panel to Show Correlation Coefficients*

Description

This function calculates the correlation between two datasets x and y and writes the textual representation into the corresponding field of the scatterplot panel. Depending on the derived value different font sizes and color schemes are applied.

Usage

```
util.panel.cor(x, y, digits = 2, cex.cor, ...)
```

Arguments

`x`	first dataset
`y`	second dataset
`digits`	number of digits after the decimal separator
`cex.cor`	desired font/text size
`...`	further arguments

util.remove.fileextension

Remove File-Extension from Filepath

Description

This function takes a filepath including the extension, removes the extension and returns the remaining path.

Usage

```
util.remove.fileextension(filePath)
```

Arguments

`filePath`	`character` string representing a filepath like `folder/file.ext`.

Details

This function accepts a filepath of the structure `folder/file.ext`. It determines the last point in the character string via a RegEx operation. The underlying logic is that most file extensions will be separated from the remaining path by that point. Then only the part of the filepath up to that point is extracted and kept. So the resulting filepath should look like `folder/file`.

Value

`character` string representing path in parameter `filePath` without the fileextension.

Examples

```
# Input is 'data.csv' and output should be 'data'.
KaggleHouse:::util.remove.fileextension("data.csv")
```

`util.write_csv` *Write data to a CSV-file.*

Description

Wrapper function around `write.table` that writes a given set of data to a file with the name `name`. Predefined separator is `;` and the used file-encoding is `utf-8`.

Usage

```
util.write_csv(data, name, ...)
```

Arguments

`data`	being a matrix or data-frame containing the writable data
`name`	being a string providing the filename
`...`	additional arguments to write.csv2

Index