Package ‘helda’

January 6, 2021

Title Preprocess Data and Get Better Insights from Machine Learning Models

Version 1.1.5

Description The main focus is on preprocessing and data visualization of machine learning models performances. Some functions allow to fill in gaps in time series using linear interpolation on panel data, some functions permit to draw lift effect and lift curve in order to benchmark machine learning models or you can even find the optimal number of clusters in agglomerative clustering algorithm.

Depends R (>= 3.5.0),
Imports dplyr (>= 0.7.8), ggplot2 (>= 3.2.0), sqldf (>= 0.4-11), stringr (>= 1.3.1), rlang (>= 0.4.2), stats (>= 3.5.0)
Suggests devtools (>= 2.2.1), testthat (>= 2.1.0), covr (>= 3.4.0)

License GPL-3
Encoding UTF-8
LazyData true

URL https://github.com/Redcart/helda

BugReports https://github.com/Redcart/helda/issues

RoxygenNote 7.1.1

Author Simon Corde [aut, cre]
Maintainer Simon Corde <simon.corde@hotmail.fr>

Repository CRAN

Date/Publication 2021-01-06 11:00:16 UTC

R topics documented:

- cluster_centroid ........................................... 2
- compute_global_inertia .................................. 3
cluster_centroid

Centroid of a cluster

Description
This function allows to compute the centroid of a cluster in a R data frame

Usage
cluster_centroid(i, data, cluster_variable)

Arguments
i an integer that represents the cluster number.
data a R data frame (all columns are required to be numeric types).
cluster_variable a character. This refers to the column name of the data frame representing the clusters.

Value
a vector of coordinates of the centroid of the cluster i.

Author(s)
Simon CORDE

References
Link to the author’s github package repository: https://github.com/Redcart/helda
Examples

```r
library(dplyr)
# We create some cluster from k-means on the iris data set
data <- iris %>% select(Sepal.Length, Sepal.Width, Petal.Length, Petal.Width)
result_kmeans <- kmeans(data, 3)
data$cluster <- result_kmeans$cluster
# We get the coordinates of the centroid of the second cluster
result <- cluster_centroid(i = 2, data = data, cluster_variable = "cluster")
result
```

---

**compute_global_inertia**

*Inertia of a data frame*

Description

This function allows to compute the inertia of a R data frame

Usage

```r
compute_global_inertia(data)
```

Arguments

data a R data frame (all columns are required to be numeric types).

Value

a numeric value representing the total inertia.

Author(s)

Simon CORDE

References

Link to the author’s github package repository: [https://github.com/Redcart/helda](https://github.com/Redcart/helda)

Examples

```r
result <- compute_global_inertia(mtcars)
result
```
compute_inertia_ahc

Intra group inertia for choosing the optimal number of clusters in Agglomerative Clustering

Description

This function allows to compute the inter group inertia from agglomerative clustering for different number of clusters

Usage

compute_inertia_ahc(data, method = "ward.D", max_clusters = 10)

Arguments

data a R data frame (all columns are required to be numeric types).
method a character. This specifies the method on which the agglomerative is built upon (by default set to "ward.D")
max_clusters an integer. The maximal number of clusters for which we intend to compute inter group inertia

Value

a vector of length max_clusters containing the inter group inertia for agglomerative clustering. The ith value of the vector corresponds to the inter group inertia from agglomerative clustering run with i clusters.

Author(s)

Simon CORDE

References

Link to the author's github package repository: https://github.com/Redcart/helda

Examples

library(dplyr)
# We select only numeric features from Iris data set
data <- iris %>% select(Sepal.Length, Sepal.Width, Petal.Length, Petal.Width)
result <- compute_inertia_ahc(data = data, max_clusters = 15)
result
create_calendar

**Description**

This function allows to create a complete empty calendar on a year scale.

**Usage**

create_calendar(data, key_variable, time_variable, start_year, end_year)

**Arguments**

- **data**: a R data frame.
- **key_variable**: a character. This represents the variable name that refers to the key variable in the panel data (an ID, ...).
- **time_variable**: a character. This represents the time variable name that permits to sort observation on a time scale.
- **start_year**: integer of the starting year of the time series.
- **end_year**: integer of the ending year of the time series.

**Value**

a R data frame with the key and time variable. Each id key is associated with all years between `start_year` and `end_year`.

**Author(s)**

Simon CORDE

**References**

Link to the author’s github package repository: [https://github.com/Redcart/helda](https://github.com/Redcart/helda)

**See Also**

`start_end_to_fill` `gap_to_fill`

**Examples**

library(dplyr)

# We take three countries from 2011 to 2018
fr_sp_ge_pop <- world_countries_pop %>% filter(country_name %in% c('France', 'Spain', 'Germany')) %>% filter(year > 2010) %>% arrange(country_name, year)
create_formula

Description
This function allows to create a formula from the columns of a data frame very quickly.

Usage
create_formula(data, position = 1)

Arguments
- data: a R data frame.
- position: integer representing the position of the column in the data frame that we want to predict. The other columns are all considered as explanatory variables.

Value
a string that contains the formula. The formula is displayed with the following format: "Y ~ X1 + X2 + ..."

Author(s)
Simon CORDE

References
Link to the author’s github package repository: https://github.com/Redcart/helda

Examples
```r
data <- iris
str(data)
result <- create_formula(data = data, position = 4)
result
```
Description

This function allows to fill intermediate gaps in panel data by linear interpolation.

Usage

gap_to_fill(data, gap_variable, key_variable, time_variable, digits = 2)

Arguments

data: a R data frame.
gap_variable: a character. This represents the name of the variable we want to fill the start and end gaps.
key_variable: a character. This represents the variable name that refers to the key variable in the panel data (an ID, ...).
time_variable: a character. This represents the time variable name that permits to sort observation on a time scale.
digits: an integer. This is the number of decimals to keep for the rounding (by default set to 2).

Value

A R data frame that contains the original columns and a new one:

- gap_variable_corrected_2: the gap variable with intermediate gaps filled

Author(s)

Simon CORDE

References

Link to the author’s github package repository: https://github.com/Redcart/helda

See Also

create_calendar start_end_to_fill
Examples

```r
clean_text <- readLines("example.txt")
```

Description

This function allows to perform k-means clustering with constrained on the size of clusters.

Usage

```r
kmeans_procedure(
  data,
  columns,
  threshold_min,
  threshold_max,
  verbose = FALSE,
  seed = 42
)
```

Arguments

- `data`: a R data frame.
- `columns`: a vector of columns names of the data frame on which we perform the kmeans algorithm. These features have to be numeric.
- `threshold_min`: an integer. It represents the minimum size for cluster.
- `threshold_max`: an integer. It represents the maximum size for cluster.
- `verbose`: a boolean. If set to TRUE print the current state of the procedure (by default set to FALSE).
- `seed`: an integer. This represents the seed for the random call (if we want the output to be reproducible).
Value

a R data frame. This contains the id of the original data frame and a column 'cluster' representing the cluster to which the observation belongs to.

Author(s)

Simon CORDE

References

Link to the author’s github package repository: https://github.com/Redcart/helda

Examples

library(dplyr)
data <- iris %>% select(Sepal.Length, Sepal.Width, Petal.Length, Petal.Width)
features <- colnames(data)
result <- kmeans_procedure(data = data, columns = features, threshold_min = 2, threshold = 10, verbose=FALSE, seed=10)

lift_curve

Lift curve graph

Description

This function allows to draw a lift curve in a ggplot style for binary classification model

Usage

lift_curve(predictions, true_labels, positive_label)

Arguments

predictions a vector of predictions. These are generally the result of a machine learning model. The predictions must be probabilities (a real number between 0 and 1).
true_labels a vector of true labels.
positive_label a character or integer that specify the positive label (Y=1) in the ‘true_labels’.

Value

a ggplot object containing the lift curve.

Author(s)

Simon CORDE
References

Link to the author’s github package repository: https://github.com/Redcart/helda

See Also

lift_effect

Examples

data_training <- titanic_training
data_validation <- titanic_validation
model_glm <- glm(formula = "Survived ~ Pclass + Sex + Age + SibSp + Fare + Embarked",
data = data_training,
family = binomial(link = "logit"))
predictions <- predict(object = model_glm, newdata = data_validation, type = "response")
plot <- lift_curve(predictions = predictions, true_labels = data_validation$Survived,
positive_label = 1)
plot

lift_effect

Description

This function allows to draw the lift effect on a graph for binary classification model

Usage

lift_effect(predictions, true_labels, positive_label)

Arguments

predictions a vector of predictions. These are generally the result of a machine learning
model. The predictions must be probabilities (a real number between 0 and 1).
true_labels a vector of true labels.
positive_label a character or integer that specify the positive label (Y=1) in the ’true_labels’.

Value

a ggplot object containing the lift effect.

Author(s)

Simon CORDE

References

Link to the author’s github package repository: https://github.com/Redcart/helda
proc_freq

See Also

lift_curve

Examples

data_training <- titanic_training
data_validation <- titanic_validation
model_glm <- glm(formula = "Survived ~ Pclass + Sex + Age + SibSp + Fare + Embarked",
data = data_training,
family = binomial(link = "logit"))
predictions <- predict(object = model_glm, newdata = data_validation, type = "response")
plot <- lift_effect(predictions = predictions, true_labels = data_validation$Survived,
positive_label = 1)
plot

proc_freq

SAS proc freq in R

Description

This function permits to reproduce the output of the SAS proc freq

Usage

proc_freq(variable, digits = 4)

Arguments

variable vector on which we want to apply the function.
digits integer that specifies the number of decimals we want to keep in the rounded figures.

Value

a R data frame of dimension [number of categories x 5]. The five columns display the following information:

- Category: different categories of the original categorical variable
- Frequency
- Percentage
- Cumulative.Frequency
- Cumulative.Percentage

Author(s)

Simon CORDE
References

Link to the author's github package repository: https://github.com/Redcart/helda

Examples

data <- iris
str(data)
result <- proc_freq(data$Species)
result

```
start_end_to_fill  Function for filling start and end gaps in time series
```

Description

This function allows to fill the start and end gaps of a time series by doing repetition of next (for the start) and previous values (for the end)

Usage

```
start_end_to_fill(data, calendar, gap_variable, key_variable, time_variable)
```

Arguments

data  a R data frame
calendar  a R data frame containing a complete empty calendar (as one can performs with create_calendar_day)
gap_variable  a character. This represents the name of the variable we want to fill the start and end gaps
key_variable  a character. This represents the variable name that refers to the key variable in the panel data (an ID, ...)
time_variable  a character. This represents the time variable name that permits to sort observation on a time scale

Value

a R data frame that contains the original columns and a new one:

- gap_variable_corrected_1: the gap variable with starts and ends filled

Author(s)

Simon CORDE

References

Link to the author’s github package repository: https://github.com/Redcart/helda
### Examples

```r
library(dplyr)

# We take three countries from 2011 to 2018
fr_sp_ge_pop <- world_countries_pop %>%
  filter(country_name %in% c('France', 'Spain', 'Germany')) %>%
  filter(year > 2010) %>%
  arrange(country_name, year)

# We artificially create some gaps in time series
fr_sp_ge_pop$population[c(1, 5, 11, 12, 24)] <- NA
fr_sp_ge_pop <- na.omit(fr_sp_ge_pop)

data_1 <- create_calendar(data = fr_sp_ge_pop, key_variable = "country_code",
  time_variable = "year", start_year = 2011, end_year = 2018)
data_2 <- start_end_to_fill(data = fr_sp_ge_pop, calendar = data_1, gap_variable = "population",
  key_variable = "country_code", time_variable = "year")
data_3 <- gap_to_fill(data = data_2, gap_variable = "population_corrected_1",
  key_variable = "country_code", time_variable = "year", digits = 1)
```

---

### Titanic testing data set

**Titanic testing data set**

**Description**

Titanic testing data set

**Usage**

titanic_testing

**Format**

Data frame of 418 observations and 11 features:

- **PassengerId**  id of the passenger
- **Pclass**  passenger class on the boat
- **Name**  name of the passenger
- **Sex**  male / female
- **Age**  age of the passenger
- **SibSp**  number of siblings/spouses aboard
- **Parch**  number of parents/children aboard
<table>
<thead>
<tr>
<th>Ticket</th>
<th>ticket no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fare</td>
<td>price of the ticket</td>
</tr>
<tr>
<td>Cabin</td>
<td>location of the cabin on the boat</td>
</tr>
<tr>
<td>Embarked</td>
<td>harbor city of boarding</td>
</tr>
</tbody>
</table>

**Source**

Kaggle Titanic Competition: [https://www.kaggle.com/c/titanic/data](https://www.kaggle.com/c/titanic/data)

**Examples**

```r
data(titanic_testing)
```

---

**titanic_training**  
*Titanic training data set*

**Description**

Titanic training data set

**Usage**

titanic_training

**Format**

Data frame of 712 observations and 12 features:

- **PassengerId**  id of the passenger
- **Survived**  dummy variable (0 if the passenger died / 1 if the passenger survived)
- **Pclass**  passenger class on the boat
- **Name**  name of the passenger
- **Sex**  male / female
- **Age**  age of the passenger
- **SibSp**  number of siblings/spouses aboard
- **Parch**  number of parents/children aboard
- **Ticket**  ticket no
- **Fare**  price of the ticket
- **Cabin**  location of the cabin on the boat
- **Embarked**  harbor city of boarding

**Source**

Kaggle Titanic Competition: [https://www.kaggle.com/c/titanic/data](https://www.kaggle.com/c/titanic/data)
Examples
data(titanic_training)

titanic_validation  Titanic validation data set

Description
Titanic validation data set

Usage
titanic_validation

Format
Data frame of 179 observations and 12 features:
  PassengerId  id of the passenger
  Survived     dummy variable (0 if the passenger died / 1 if the passenger survived)
  Pclass       passenger class on the boat
  Name         name of the passenger
  Sex          male / female
  Age          age of the passenger
  SibSp        number of siblings/spouses aboard
  Parch        number of parents/children aboard
  Ticket       ticket no
  Fare         price of the ticket
  Cabin        location of the cabin on the boat
  Embarked     harbor city of boarding

Source
Kaggle Titanic Competition: https://www.kaggle.com/c/titanic/data

Examples
data(titanic_validation)
windows_to_linux_path  Convert windows path into linux path

Description
This function allows to make conversion of windows path into linux path.

Usage
windows_to_linux_path()

Details
When the function is called, a prompt asks for the windows path to be converted in the R console. Enter a windows path or copy paste one. Then type ENTER. The Linux converted path appears.

Value
None.

Author(s)
Simon CORDE

References
Link to the author’s github package repository: https://github.com/Redcart/helda

world_countries_pop  World countries population from 1960 to 2018

Description
World countries population from 1960 to 2018

Usage
world_countries_pop

Format
R Data frame with 15576 observations and 4 variables

- **country_name**  name of the country
- **country_code**  code of the country in three letters
- **year**  year from 1960 to 2018
- **population**  number of people
Source

World Bank website https://data.worldbank.org/indicator/SP.POP.TOTL

Examples

data(world_countries_pop)
Index

* SAS
  proc_freq, 11
* agglomerative
  compute_inertia_ahc, 4
* calendar
  create_calendar, 5
* categorical
  proc_freq, 11
* centroids
  cluster_centroid, 2
* classification
  lift_curve, 9
  lift_effect, 10
* clustering
  compute_global_inertia, 3
  compute_inertia_ahc, 4
  create_formula, 6
* frequency
  proc_freq, 11
* gaps
  create_calendar, 5
  gap_to_fill, 7
  start_end_to_fill, 12
* inertia
  compute_global_inertia, 3
  compute_inertia_ahc, 4
* interpolation
  gap_to_fill, 7
* kmeans
  kmeans_procedure, 8
* learning
  lift_curve, 9
  lift_effect, 10
* lift
  lift_curve, 9
  lift_effect, 10
* linux
  windows_to_linux_path, 16
* machine
  lift_curve, 9
  lift_effect, 10
* path
  windows_to_linux_path, 16
* proc_freq
  proc_freq, 11
* series
  create_calendar, 5
  gap_to_fill, 7
  start_end_to_fill, 12
* sizes
  kmeans_procedure, 8
* table
  proc_freq, 11
INDEX

* time
  create_calendar, 5
  gap_to_fill, 7
  start_end_to_fill, 12
* variable
  proc_freq, 11
* windows
  windows_to_linux_path, 16

cluster_centroid, 2
compute_global_inertia, 3
compute_inertia_ahc, 4
create_calendar, 5, 7, 13
create_formula, 6
gap_to_fill, 5, 7, 13
kmeans_procedure, 8
lift_curve, 9
lift_effect, 10
proc_freq, 11
start_end_to_fill, 5, 7, 12
titanic_testing, 13
titanic_training, 14
titanic_validation, 15
windows_to_linux_path, 16
world_countries_pop, 16