Package 'anomalyDetection'

March 7, 2018

Type Package

Title Implementation of Augmented Network Log Anomaly Detection Procedures

Version 0.2.5

Maintainer Bradley Boehmke <bradleyboehmke@gmail.com>

Date 2018-02-24

Description Implements procedures developed by Gutierrez et al. (2017, <https://journal.r-project.org/archive/2017/RJ-2017-039/index.html>) to aid in detecting network log anomalies. By combining various multivariate analytic approaches relevant to network anomaly detection, it provides cyber analysts efficient means to detect suspected anomalies requiring further evaluation.

URL https://github.com/koalaverse/anomalyDetection

BugReports https://github.com/koalaverse/anomalyDetection/issues

License GPL (>= 2)

Encoding UTF-8

LazyData true

Depends R (>= 2.10)

Imports caret, dplyr, ggplot2, gmp, magrittr, MASS, plyr, purrr, Rcpp (>= 0.12.11), stats, tibble, tidyr,

RoxygenNote 6.0.1

Suggests gplots, knitr, RColorBrewer, rmarkdown, testthat,

LinkingTo Rcpp, RcppArmadillo

VignetteBuilder knitr

NeedsCompilation yes

Author Bradley Boehmke [aut, cre],
Brandon Greenwell [aut],
Jason Freels [aut],
Robert Gutierrez [aut]

Repository CRAN

Date/Publication 2018-03-07 16:18:12 UTC

R topics documented:

`anomalyDetection` *anomalyDetection: An R package for implementing augmented network log anomoly detection procedures.*

Description

anomalyDetection: An R package for implementing augmented network log anomoly detection procedures.

`bd_row` *Breakdown for Mahalanobis Distance*

Description

`bd_row` indicates which variables in data are driving the Mahalanobis distance for a specific row `r`, relative to the mean vector of the data.

Usage

```
bd_row(data, row, n = NULL)
```

Arguments

data	numeric data
row	row of interest
n	number of values to return. By default, will return all variables (columns) with their respective differences. However, you can choose to view only the top n variables by setting the n value.

Value

Returns a vector indicating the variables in data that are driving the Mahalanobis distance for the respective row.

See Also

mahalanobis_distance for computing the Mahalanobis Distance values

Examples

```
## Not run:
x = matrix(rnorm(200*3), ncol = 10)
colnames(x) = paste0("C", 1:ncol(x))

# compute the relative differences for row 5 and return all variables
x %>%
  mahalanobis_distance("bd", normalize = TRUE) %>%
  bd_row(5)

# compute the relative differences for row 5 and return the top 3 variables
# that are influencing the Mahalanobis Distance the most
x %>%
  mahalanobis_distance("bd", normalize = TRUE) %>%
  bd_row(5, 3)

## End(Not run)
```

---

factor_analysis *Factor Analysis with Varimax Rotation*

Description

factor_analysis reduces the structure of the data by relating the correlation between variables to a set of factors, using the eigen-decomposition of the correlation matrix.

Usage

```
factor_analysis(data, hc_points)
```

Arguments

data	numeric data
hc_points	vector of eigenvalues [designed to use output from horns_curve]

Value

A list containing:

1. fa_loadings: numerical matrix with the original factor loadings
2. fa_scores: numerical matrix with the row scores for each factor
3. fa_loadings_rotated: numerical matrix with the varimax rotated factor loadings
4. fa_scores_rotated: numerical matrix with the row scores for each varimax rotated factor
5. num_factors: numeric vector identifying the number of factors

References

H. F. Kaiser, "The Application of Electronic Computers to Factor Analysis," Educational and Psychological Measurement, 1960.

See Also

horns_curve for computing the average eigenvalues used for hc_points argument

Examples

```
# Perform Factor Analysis with matrix \code{x}
x <- matrix(rnorm(200*3), ncol = 10)

x %>%
  horns_curve() %>%
  factor_analysis(x, hc_points = .)
```

---

factor_analysis_results

Easy Access to Factor Analysis Results

Description

factor_analysis_result Provides easy access to factor analysis results

Usage

```
factor_analysis_results(data, results = 1)
```

Arguments

data	list output from factor_analysis
results	factor analysis results to extract. Can use either results name or number (i.e. fa_scores or 2)::

1. fa_loadings (default)
2. fa_scores
3. fa_loadings_rotated
4. fa_scores_rotated
5. num_factors

Value

Returns the one of the selected results:

1. fa_loadings: numerical matrix with the original factor loadings
2. fa_scores: numerical matrix with the row scores for each factor
3. fa_loadings_rotated: numerical matrix with the varimax rotated factor loadings
4. fa_scores_rotated: numerical matrix with the row scores for each varimax rotated factor
5. num_factors: numeric vector identifying the number of factors

See Also

factor_analysis for computing the factor analysis results

Examples

```
# An efficient means for getting factor analysis results
x <- matrix(rnorm(200*3), ncol = 10)
N <- nrow(x)
p <- ncol(x)

x %>%
  horns_curve() %>%
  factor_analysis(x, hc_points = .) %>%
  factor_analysis_results(fa_scores_rotated)
```

---

get_all_factors *Find All Factors*

Description

get_all_factors finds all factor pairs for a given integer (i.e. a number that divides evenly into another number).

Usage

```
get_all_factors(n)
```

Arguments

n	number to be factored

Value

A list containing the integer vector(s) containing all factors for the given n inputs.

Source

http://stackoverflow.com/a/6425597/3851274

Examples

```
# Find all the factors of 39304
get_all_factors(39304)
```

---

hmat *Plot a Histogram Matrix*

Description

Display a histogram matrix for visual inspection of anomalous observation detection. The color of the blocks represents how anomalous each block is, where a lighter blue represents a more anomalous block. The size of the points indicate which values are driving the anomaly, with larger blocks representing more anomalous values.

Usage

```
hmat(data, input = "data", top = 20, order = "numeric",
  block_length = NULL, level_limit = 50, level_keep = 10,
  partial_block = TRUE, na.rm = FALSE, min_var = 0.1, max_cor = 0.9,
  action = "exclude", output = "both", normalize = FALSE)
```

Arguments

data	the data set (data frame or matrix)
input	the type of input data being passed to the function. data for a raw categorical data set, SV for a state vector input, and MD if the input has already had the Mahalanobis distances calculated
top	how many of the most anomalous blocks you would like to display (default 20)

order	whether to show the anomalous blocks in numeric order or in order of most anomalous to least anomalous (default is "numeric", other choice is "anomaly")
block_length	argument fed into tabulate_state_vector, necessary if input = data
level_limit	argument fed into tabulate_state_vector, if the number of unique categories for a variable exceeds this number, only keep a limited number of the most popular values (default 50)
level_keep	argument fed into tabulate_state_vector, if level_limit is exceeded, keep this many of the most popular values (default 10)
partial_block	argument fed into tabulate_state_vector, if the number of entries is not divisible by the block_length, this logical decides whether to keep the smaller last block (default TRUE)
na.rm	whether to keep track of missing values as part of the analysis or ignore them (default FALSE)
min_var	argument fed into mc_adjust, if a column in the state vector has variance less than this value, remove it (default 0.1)
max_cor	argument fed into mc_adjust, if a column in the state vector has correlation greater than this value, remove it (default 0.9)
action	argument fed into mc_adjust, if a column does not fall in the specified range, determine what to do with it (default "exclude")
output	argument fed into mahalanobis_distance that decides whether to add a column for the Mahalanobis Distance ('MD'), the breakdown distances ('BD') or both (default "both")
normalize	argument fed into mahalanobis_distance that decides whether to normalize the values by column (default = FALSE)

Examples

```
## Not run:
# Data set input
hmat(security_logs,block_length = 8)

# Data Set input with top 10 blocks displayed
hmat(security_logs, top = 10, block_length = 5)

# State Vector Input
tabulate_state_vector(security_logs, block_length = 6, level_limit = 20) %>%
  hmat(input = "SV")

## End(Not run)
```

`horns_curve` *Horn's Parallel Analysis*

Description

Computes the average eigenvalues produced by a Monte Carlo simulation that randomly generates a large number of nxp matrices of standard normal deviates.

Usage

```
horns_curve(data, n, p, nsim = 1000L)
```

Arguments

`data`	A matrix or data frame.
`n`	Integer specifying the number of rows.
`p`	Integer specifying the number of columns.
`nsim`	Integer specifying the number of Monte Carlo simulations to run. Default is `1000`.

Value

A vector of length p containing the averaged eigenvalues. The values can then be plotted or compared to the true eigenvalues from a dataset for a dimensionality reduction assessment.

References

J. L. Horn, "A rationale and test for the number of factors in factor analysis," Psychometrika, vol. 30, no. 2, pp. 179-185, 1965.

Examples

```
# Perform Horn's Parallel analysis with matrix n x p dimensions
x <- matrix(rnorm(200 * 10), ncol = 10)
horns_curve(x)
horns_curve(n = 200, p = 10)
plot(horns_curve(x))  # scree plot
```

inspect_block *Block Inspection*

Description

inspect_block creates a list where the original data has been divided into blocks denoted in the state vector. Streamlines the process of inspecting specific blocks of interest.

Usage

```
inspect_block(data, block_length)
```

Arguments

data	data
block_length	integer value to divide data

Value

A list where each item is a data frame that contains the original data for each block denoted in the state vector.

See Also

tabulate_state_vector for creating the state vector matrix based on desired blocks.

Examples

```
inspect_block(security_logs, 30)
```

kaisers_index *Kaiser's Index of Factorial Simplicity*

Description

kaisers_index computes scores designed to assess the quality of a factor analysis solution. It measures the tendency towards unifactoriality for both a given row and the entire matrix as a whole. Kaiser proposed the evaluations of the score shown below:

1. In the .90s: Marvelous
2. In the .80s: Meritorious
3. In the .70s: Middling
4. In the .60s: Mediocre

5. In the .50s: Miserable
6. < .50: Unacceptable

Use as basis for selecting original or rotated loadings/scores in `factor_analysis`.

Usage

```
kaisers_index(loadings)
```

Arguments

`loadings`	numerical matrix of the factor loadings

Value

Vector containing the computed score

References

H. F. Kaiser, "An index of factorial simplicity," Psychometrika, vol. 39, no. 1, pp. 31-36, 1974.

See Also

`factor_analysis` for computing the factor analysis loadings

Examples

```
# Perform Factor Analysis with matrix \code{x}
x <- matrix(rnorm(200*3), ncol = 10)

x %>%
  horns_curve() %>%
  factor_analysis(x, hc_points = .) %>%
  factor_analysis_results(fa_loadings_rotated) %>%
  kaisers_index()
```

`mahalanobis_distance` *Mahalanobis Distance*

Description

Calculates the distance between the elements in a data set and the mean vector of the data for outlier detection. Values are independent of the scale between variables.

Usage

```
mahalanobis_distance(data, output = c("md", "bd", "both"),
  normalize = FALSE)

## S3 method for class 'matrix'
mahalanobis_distance(data, output = c("md", "bd", "both"),
  normalize = FALSE)

## S3 method for class 'data.frame'
mahalanobis_distance(data, output = c("md", "bd",
  "both"), normalize = FALSE)
```

Arguments

data	A matrix or data frame. Data frames will be converted to matrices via data.matrix.
output	Character string specifying which distance metric(s) to compute. Current options include: "md" for Mahalanobis distance (default); "bd" for absolute breakdown distance (used to see which columns drive the Mahalanobis distance); and "both" to return both distance metrics.
normalize	Logical indicating whether or not to normalize the breakdown distances within each column (so that breakdown distances across columns can be compared).

Value

If output = "md", then a vector containing the Mahalanobis distances is returned. Otherwise, a matrix.

References

W. Wang and R. Battiti, "Identifying Intrusions in Computer Networks with Principal Component Analysis," in First International Conference on Availability, Reliability and Security, 2006.

Examples

```
## Not run:
# Simulate some data
x <- data.frame(C1 = rnorm(100), C2 = rnorm(100), C3 = rnorm(100))

# Add Mahalanobis distances
x %>% dplyr::mutate(MD = mahalanobis_distance(x))

# Add Mahalanobis and breakdown distances
x %>% cbind(mahalanobis_distance(x, output = "both"))

# Add Mahalanobis and normalized breakdown distances
x %>% cbind(mahalanobis_distance(x, output = "both", normalize = TRUE))

## End(Not run)
```

mc_adjust *Multi-Collinearity Adjustment*

Description

mc_adjust handles issues with multi-collinearity.

Usage

```
mc_adjust(data, min_var = 0.1, max_cor = 0.9, action = "exclude")
```

Arguments

data	named numeric data object (either data frame or matrix)
min_var	numeric value between 0-1 for the minimum acceptable variance (default = 0.1)
max_cor	numeric value between 0-1 for the maximum acceptable correlation (default = 0.9)
action	select action for handling columns causing multi-collinearity issues 1. exclude: exclude all columns causing multi-collinearity issues (default) 2. select: identify the columns causing multi-collinearity issues and allow the user to interactively select those columns to remove

Details

mc_adjust handles issues with multi-collinearity by first removing any columns whose variance is close to or less than min_var. Then, it removes linearly dependent columns. Finally, it removes any columns that have a high absolute correlation value equal to or greater than max_cor.

Value

mc_adjust returns the numeric data object supplied minus variables violating the minimum acceptable variance (min_var) and the maximum acceptable correlation (max_cor) levels.

Examples

```
## Not run:
x <- matrix(runif(100), ncol = 10)
x %>%
  mc_adjust()

x %>%
  mc_adjust(min_var = .15, max_cor = .75, action = "select")

## End(Not run)
```

`principal_components` *Principal Component Analysis*

Description

`principal_components` relates the data to a set of a components through the eigen-decomposition of the correlation matrix, where each component explains some variance of the data and returns the results as an object of class prcomp.

Usage

```
principal_components(data, retx = TRUE, center = TRUE, scale. = FALSE,
  tol = NULL, ...)
```

Arguments

`data`	numeric data.
`retx`	a logical value indicating whether the rotated variables should be returned.
`center`	a logical value indicating whether the variables should be shifted to be zero centered. Alternately, a vector of length equal the number of columns of x can be supplied. The value is passed to scale.
`scale.`	a logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place. The default is FALSE for consistency with S, but in general scaling is advisable. Alternatively, a vector of length equal the number of columns of `data` can be supplied. The value is passed to scale.
`tol`	a value indicating the magnitude below which components should be omitted. (Components are omitted if their standard deviations are less than or equal to tol times the standard deviation of the first component.) With the default null setting, no components are omitted. Other settings for tol could be `tol = 0` or `tol = sqrt(.Machine$double.eps)`, which would omit essentially constant components.
`...`	arguments passed to or from other methods.

Details

The calculation is done by a singular value decomposition of the (centered and possibly scaled) data matrix, not by using eigen on the covariance matrix. This is generally the preferred method for numerical accuracy

Value

`principal_components` returns a list containing the following components:

1. `pca_sdev`: the standard deviations of the principal components (i.e., the square roots of the eigenvalues of the correlation matrix, though the calculation is actually done with the singular values of the data matrix).

2. pca_loadings: the matrix of variable loadings (i.e., a matrix whose columns contain the eigenvectors).
3. pca_rotated: if retx is TRUE the value of the rotated data (the centred (and scaled if requested) data multiplied by the rotation matrix) is returned. Hence, cov(x) is the diagonal matrix diag(sdev^2).
4. pca_center: the centering used
5. pca_scale: whether scaling was used

See Also

prcomp, biplot.prcomp, screeplot, cor, cov, svd, eigen

Examples

```
x <- matrix(rnorm(200 * 3), ncol = 10)
principal_components(x)
principal_components(x, scale = TRUE)
```

---

principal_components_result

Easy Access to Principal Component Analysis Results

Description

principal_components_result Provides easy access to principal component analysis results

Usage

```
principal_components_result(data, results = 2)
```

Arguments

data	list output from principal_components
results	principal component analysis results to extract. Can use either results name or number (i.e. pca_loadings or 2): 1. pca_sdev 2. pca_loadings (default) 3. pca_rotated 4. pca_center 5. pca_scale

Value

Returns one of the selected results:

1. `pca_sdev`: the standard deviations of the principal components (i.e., the square roots of the eigenvalues of the correlation matrix, though the calculation is actually done with the singular values of the data matrix).
2. `pca_loadings`: the matrix of variable loadings (i.e., a matrix whose columns contain the eigenvectors).
3. `pca_rotated`: if `retx` is `TRUE` the value of the rotated data (the centred (and scaled if requested) data multiplied by the rotation matrix) is returned. Hence, `cov(x)` is the diagonal matrix `diag(sdev^2)`.
4. `pca_center`: the centering used
5. `pca_scale`: whether scaling was used

See Also

`principal_components` for computing the principal components results

Examples

```
# An efficient means for getting principal component analysis results
x <- matrix(rnorm(200 * 3), ncol = 10)

principal_components(x) %>%
  principal_components_result(pca_loadings)
```

---

security_logs *Security Log Data*

Description

A mock dataset containing common information that appears in security logs.

Usage

```
security_logs
```

Format

A data frame with 300 rows and 10 variables:

Device_Vendor Company who made the device

Device_Product Name of the security device

Device_Action Outcome result of access

Src_IP IP address of the source

Dst_IP IP address of the destination

Src_Port Port identifier of the source

Dst_Port Port identifier of the destination

Protocol Transport protocol used

Country_Src Country of the source

Bytes_TRF Number of bytes transferred

`tabulate_state_vector` *Tabulate State Vector*

Description

`tabulate_state_vector` employs a tabulated vector approach to transform security log data into unique counts of data attributes based on time blocks. Taking a contingency table approach, this function separates variables of type character or factor into their unique levels and counts the number of occurrences for those levels within each block. Due to the large number of unique IP addresses, this function allows for the user to determine how many IP addresses they would like to investigate. The function tabulates the most popular IP addresses.

Usage

```
tabulate_state_vector(data, block_length, level_limit = 50L,
  level_keep = 10L, partial_block = FALSE, na.rm = FALSE)
```

Arguments

`data`	data
`block_length`	integer value to divide data by
`level_limit`	integer value to determine the cutoff for the number of factors in a column to display before being reduced to show the number of levels to keep (default is 50)
`level_keep`	integer value indicating the top number of factor levels to retain if a column has more than the level limit (default is 10)
`partial_block`	a logical which determines whether incomplete blocks are kept in the analysis in the case where the number of log entries isn't evenly divisible by the `block_length`
`na.rm`	whether to keep track of missing values as part of the analysis or ignore them

Value

A data frame where each row represents one block and the columns count the number of occurrences that character/factor level occurred in that block

Examples

```
tabulate_state_vector(security_logs, 30)
```

`%>%` *Pipe functions*

Description

Like dplyr, anomalyDetection also uses the pipe function, `%>%` to turn function composition into a series of imperative statements.

Arguments

`lhs, rhs`	An R object and a function to apply to it

Examples

```
x <- matrix(rnorm(200*3), ncol = 10)
N <- nrow(x)
p <- ncol(x)

# Instead of
hc <- horns_curve(x)
fa <- factor_analysis(x, hc_points = hc)
factor_analysis_results(fa, fa_scores_rotated)

# You can write
horns_curve(x) %>%
  factor_analysis(x, hc_points = .) %>%
  factor_analysis_results(fa_scores_rotated)
```

Index