

# Package ‘arlclustering’

May 7, 2026

**Type** Package

**Title** Exploring Social Network Structures Through Friendship-Driven  
Community Detection with Association Rules Mining

**Version** 1.0.5

**Maintainer** Mohamed El-Moussaoui <med.elmoussaoui.ced@gmail.com>

**Description** Implements an innovative approach to community detection in social networks using Association Rules Learning. The package provides tools for processing graph and rules objects, generating association rules, and detecting communities based on node interactions. Designed to facilitate advanced research in Social Network Analysis, this package leverages association rules learning for enhanced community detection. This approach is described in El-Moussaoui et al. (2021) <[doi:10.1007/978-3-030-66840-2\\_3](https://doi.org/10.1007/978-3-030-66840-2_3)>.

**License** GPL-3

**URL** <https://github.com/assuom44/arlclustering>

**BugReports** <https://github.com/assuom44/arlclustering/issues>

**LazyLoad** false

**Encoding** UTF-8

**RoxygenNote** 7.2.3

**Suggests** testthat (>= 3.0.0), knitr, rmarkdown, tidy

**Imports** methods, igraph, arules, grDevices, graphics, stats

**Config/testthat/edition** 3

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Mohamed El-Moussaoui [aut, cre],  
Mohamed Hanine [aut],  
Ali Kartit [ths],  
Tarik Agouti [rev]

**Repository** CRAN

**Date/Publication** 2024-09-11 15:00:02 UTC

## Contents

arlc_calculate_mode . . . . .	2
arlc_clean_final_rules . . . . .	3
arlc_clusters_plot . . . . .	4
arlc_convert_date_format . . . . .	5
arlc_count_na . . . . .	5
arlc_df_summary . . . . .	6
arlc_fct_clean_transactions . . . . .	6
arlc_fct_get_best_apriori_thresholds . . . . .	7
arlc_file_exists_readable . . . . .	8
arlc_generate_clusters . . . . .	9
arlc_generate_date_sequence . . . . .	10
arlc_generate_uid . . . . .	10
arlc_gen_gross_rules . . . . .	11
arlc_gen_transactions . . . . .	12
arlc_get_apriori_thresholds . . . . .	12
arlc_get_network_dataset . . . . .	13
arlc_get_NonR_rules . . . . .	14
arlc_get_significant_rules . . . . .	15
arlc_is_numeric_vector . . . . .	16
arlc_list_to_df . . . . .	17
arlc_measure_time . . . . .	17
arlc_normalize_vector . . . . .	18
arlc_replace_na . . . . .	18
<b>Index</b>	<b>19</b>

---

arlc\_calculate\_mode     *Calculate the Mode of a Vector*

---

### Description

This function calculates the mode of a vector.

### Usage

```
arlc_calculate_mode(x)
```

### Arguments

x                     A vector.

### Value

The mode of the vector.

**Examples**

```
arlc_calculate_mode(c(1, 2, 2, 3, 4))
```

---

```
arlc_clean_final_rules
```

*Clean Final Rules*

---

**Description**

This function cleans the final set of association rules.

**Usage**

```
arlc_clean_final_rules(final_rules)
```

**Arguments**

`final_rules` A set of final rules to be cleaned.

**Value**

A cleaned set of rules.

**Examples**

```
library(arlclustering)
# Create a sample transactions dataset
sample_gml_file <- system.file("extdata", "karate.gml", package = "arlclustering")
g <- arlc_get_network_dataset(sample_gml_file, "Karate Club")
trans <- arlc_gen_transactions(g$graph)
supportRange <- seq(0.1, 0.2, by = 0.1)
Conf <- 0.5
params <- arlc_get_apriori_thresholds(trans, supportRange, Conf)
grossRules <- arlc_gen_gross_rules(trans, params$minSupp, params$minConf, 1, params$lenRules)
nonRR_rules <- arlc_get_NonR_rules(grossRules$GrossRules)
NonRRSig_rules <- arlc_get_significant_rules(trans, nonRR_rules$FiltredRules)
cleaned_rules <- arlc_clean_final_rules(NonRRSig_rules$FiltredRules)
message(cleaned_rules)
```

---

arlc\_clusters\_plot      *Plot Graph with Custom Layout and Communities*

---

## Description

This function plots a graph with specified aesthetics and highlights communities.

## Usage

```
arlc_clusters_plot(g, graphLabel, clusters)
```

## Arguments

<code>g</code>	An igraph object representing the graph.
<code>graphLabel</code>	A character string for the graph label to be displayed in the title.
<code>clusters</code>	A list of clusters to highlight in the plot.

## Value

The function produces a plot as a side effect.

## Examples

```
library(arlclustering)
# Create a sample transactions dataset
sample_gml_file <- system.file("extdata", "karate.gml", package = "arlclustering")
g <- arlc_get_network_dataset(sample_gml_file, "Karate Club")
trans <- arlc_gen_transactions(g$graph)
supportRange <- seq(0.1, 0.2, by = 0.1)
Conf <- 0.5
params <- arlc_get_apriori_thresholds(trans, supportRange, Conf)
grossRules <- arlc_gen_gross_rules(trans, params$minSupp, params$minConf, 1, params$lenRules)
nonRR_rules <- arlc_get_NonR_rules(grossRules$GrossRules)
NonRRSig_rules <- arlc_get_significant_rules(trans, nonRR_rules$FiltredRules)
cleaned_rules <- arlc_clean_final_rules(NonRRSig_rules$FiltredRules)
clusters <- arlc_generate_clusters(cleaned_rules)
arlc_clusters_plot(g$graph, "Karate Club", clusters$Clusters)
```

---

`arlc_convert_date_format`*Convert a Date to a Different Format*

---

**Description**

This function converts a date from one format to another.

**Usage**

```
arlc_convert_date_format(date_str, from_format, to_format)
```

**Arguments**

<code>date_str</code>	A date string.
<code>from_format</code>	The current format of the date string.
<code>to_format</code>	The desired format of the date string.

**Value**

The date string in the new format.

**Examples**

```
arlc_convert_date_format("2023-01-01", "%Y-%m-%d", "%d-%m-%Y")
```

---

`arlc_count_na`*Count NA Values in a Data Frame*

---

**Description**

This function counts the number of NA values in each column of a data frame.

**Usage**

```
arlc_count_na(df)
```

**Arguments**

<code>df</code>	A data frame.
-----------------	---------------

**Value**

A named vector with the count of NA values in each column.

**Examples**

```
arlc_count_na(data.frame(a = c(1, NA, 3), b = c(NA, NA, 3)))
```

arlc\_df\_summary

*Create a Summary of a Data Frame*

---

**Description**

This function creates a summary of a data frame including count, mean, median, and standard deviation for numeric columns.

**Usage**

```
arlc_df_summary(df)
```

**Arguments**

df                    A data frame.

**Value**

A data frame summarizing the statistics of the input data frame.

**Examples**

```
arlc_df_summary(data.frame(a = c(1, 2, 3, 4, 5), b = c(5, 4, 3, 2, 1)))
```

---

arlc\_fct\_clean\_transactions

*Clean Transactions by Removing Overlapping Sets*

---

**Description**

This function processes a list of sets and removes those that are fully overlapped by other sets.

**Usage**

```
arlc_fct_clean_transactions(all_sets)
```

**Arguments**

all\_sets             A list of sets where each set is a vector of elements.

**Details**

The function iterates through each set and checks if it is fully overlapped by any other set. If a set is fully overlapped, it is excluded from the final list of sets. The result is a list of sets with no fully overlapped sets.

**Value**

A list of sets with fully overlapped sets removed.

**Examples**

```
library(arlclustering)
# Create a sample transactions dataset
sample_gml_file <- system.file("extdata", "karate.gml", package = "arlclustering")
g <- arlc_get_network_dataset(sample_gml_file, "Karate Club")
trans <- arlc_gen_transactions(g$graph)
supportRange <- seq(0.1, 0.2, by = 0.1)
Conf <- 0.5
params <- arlc_get_apriori_thresholds(trans, supportRange, Conf)
grossRules <- arlc_gen_gross_rules(trans, params$minSupp, params$minConf, 1, params$lenRules)
nonRR_rules <- arlc_get_NonR_rules(grossRules$GrossRules)
NonRRSig_rules <- arlc_get_significant_rules(trans, nonRR_rules$FiltredRules)
cleaned_rules <- arlc_clean_final_rules(NonRRSig_rules$FiltredRules)

vec <- lapply(cleaned_rules, function(v) unique(unlist(v)))
vec2 <- split(vec, sapply(vec, `[`, 1))
sorted_result <- lapply(vec2, function(v) sort(unique(unlist(v))))

clusters <- arlc_fct_clean_transactions(sorted_result)

message (clusters)
```

---

```
arlc_fct_get_best_apriori_thresholds
```

*Get Best Apriori Thresholds*

---

**Description**

This function finds the best support and confidence thresholds for the Apriori algorithm to maximize the lift of the generated association rules.

**Usage**

```
arlc_fct_get_best_apriori_thresholds(transactions, support_range, conf)
```

**Arguments**

transactions	A transaction dataset of class transactions from the arules package.
support_range	A numeric vector specifying the range of support values to be tested.
conf	A numeric value (0.5 or 1.0) specifying the confidence value.

**Details**

This function iterates through the given ranges of support and confidence values, applies the Apriori algorithm to find association rules for each pair of values, and selects the pair that produces rules with the highest lift.

**Value**

A numeric vector containing the best support, best confidence, highest lift, and the number of rules found. The return value is a named vector with elements best\_support, best\_confidence, best\_lift, and len\_rules.

**Examples**

```
library(arlclustering)
sample_gml_file <- system.file("extdata", "karate.gml", package = "arlclustering")
g <- arlc_get_network_dataset(sample_gml_file, "Karate Club")
trans <- arlc_gen_transactions(g$graph)
supportRange <- seq(0.1, 0.2, by = 0.1)
Conf <- 0.5
best_thresholds <- arlc_fct_get_best_apriori_thresholds(trans, supportRange, Conf)
```

---

arlc\_file\_exists\_readable

*Check if a File Exists and is Readable*

---

**Description**

This function checks if a file exists and is readable.

**Usage**

```
arlc_file_exists_readable(filepath)
```

**Arguments**

filepath            The path to the file.

**Value**

TRUE if the file exists and is readable, FALSE otherwise.

**Examples**

```
arlc_file_exists_readable("example.txt")
```

---

`arlc_generate_clusters`*Generate Clusters*

---

## Description

This function takes a vector of preprocessed rules, combines elements starting with the same value, groups segments by the starting value, sorts elements within each segment, and returns potential clusters.

## Usage

```
arlc_generate_clusters(vec)
```

## Arguments

`vec`                    A vector of preprocessed rules.

## Details

This function generates potential clusters based on preprocessed rules.

## Value

A list of unique and potential clusters.

## Examples

```
library(arlclustering)
# Create a sample transactions dataset
sample_gml_file <- system.file("extdata", "karate.gml", package = "arlclustering")
g <- arlc_get_network_dataset(sample_gml_file, "Karate Club")
trans <- arlc_gen_transactions(g$graph)
supportRange <- seq(0.1, 0.2, by = 0.1)
Conf <- 0.5
params <- arlc_get_apriori_thresholds(trans, supportRange, Conf)
grossRules <- arlc_gen_gross_rules(trans, params$minSupp, params$minConf, 1, params$lenRules)
nonRR_rules <- arlc_get_NonR_rules(grossRules$GrossRules)
NonRRSig_rules <- arlc_get_significant_rules(trans, nonRR_rules$FiltredRules)
cleaned_rules <- arlc_clean_final_rules(NonRRSig_rules$FiltredRules)
clusters <- arlc_generate_clusters(cleaned_rules)
```

---

arlc\_generate\_date\_sequence  
*Generate a Sequence of Dates*

---

**Description**

This function generates a sequence of dates between two given dates.

**Usage**

```
arlc_generate_date_sequence(start_date, end_date, by = "day")
```

**Arguments**

start_date	The start date.
end_date	The end date.
by	The step size for the sequence (e.g., "day", "week", "month").

**Value**

A vector of dates.

**Examples**

```
arlc_generate_date_sequence("2023-01-01", "2023-01-10", "day")
```

---

arlc\_generate\_uid      *Generate a Unique Identifier*

---

**Description**

This function generates a unique identifier string.

**Usage**

```
arlc_generate_uid(length = 10)
```

**Arguments**

length	The length of the unique identifier. Default is 10.
--------	---

**Value**

A unique identifier string.

**Examples**

```
arlc_generate_uid()  
arlc_generate_uid(15)
```

---

arlc\_gen\_gross\_rules *Get Gross Rules*

---

**Description**

This function generates gross association rules from transactions.

**Usage**

```
arlc_gen_gross_rules(transactions, minSupp, minConf, minLenRules, maxLenRules)
```

**Arguments**

transactions	A transactions object.
minSupp	Minimum support threshold.
minConf	Minimum confidence threshold.
minLenRules	Minimum length of rules.
maxLenRules	Maximum length of rules.

**Value**

A set of gross association rules.

**Examples**

```
library(arlclustering)  
# Create a sample transactions dataset  
sample_gml_file <- system.file("extdata", "karate.gml", package = "arlclustering")  
g <- arlc_get_network_dataset(sample_gml_file, "Karate Club")  
trans <- arlc_gen_transactions(g$graph)  
supportRange <- seq(0.1, 0.2, by = 0.1)  
Conf <- 0.5  
params <- arlc_get_apriori_thresholds(trans, supportRange, Conf)  
grossRules <- arlc_gen_gross_rules(trans, params$minSupp, params$minConf, 1, params$lenRules)
```

---

arlc\_gen\_transactions *Get Transactional Dataset*

---

### Description

This function generates a transactional dataset from a graph.

### Usage

```
arlc_gen_transactions(graph)
```

### Arguments

graph            A graph object.

### Value

A transactional dataset.

### Examples

```
library(arlclustering)
# Create a sample transactions dataset
sample_gml_file <- system.file("extdata", "karate.gml", package = "arlclustering")
g <- arlc_get_network_dataset(sample_gml_file, "Karate Club")
trans <- arlc_gen_transactions(g$graph)
```

---

arlc\_get\_apriori\_thresholds  
*Get Apriori Thresholds*

---

### Description

This function takes a transaction dataset and ranges for support and confidence, computes the best thresholds, and returns the best minimum support, minimum confidence, best lift, total number of gross rules, and ratio of generated rules to total number of transactions.

### Usage

```
arlc_get_apriori_thresholds(trx, supportRange, Conf)
```

### Arguments

trx            A transaction dataset of class transactions from the arules package.  
supportRange   A sequence of values representing the range for minimum support.  
Conf           A sequence of values representing the range for minimum confidence.

**Details**

This function generates gross rules based on the best obtained thresholds.

This function iterates through the given ranges of support and confidence values, applies the Apriori algorithm to find association rules for each pair of values, and selects the pair that produces rules with the highest lift. The function then returns the best thresholds along with the lift, number of rules, and their ratio to the total transactions.

**Value**

A list containing:

minSupp	The best minimum support value.
minConf	The best minimum confidence value.
bestLift	The highest lift value obtained.
lenRules	The total number of gross rules generated.
ratio	The ratio of generated rules to the total number of transactions.

**Examples**

```
library(arlclustering)
# Create a sample transactions dataset
sample_gml_file <- system.file("extdata", "karate.gml", package = "arlclustering")
g <- arlc_get_network_dataset(sample_gml_file, "Karate Club")
trans <- arlc_gen_transactions(g$graph)
supportRange <- seq(0.1, 0.2, by = 0.1)
Conf <- 0.5
params <- arlc_get_apriori_thresholds(trans, supportRange, Conf)
message(params$minSupp)
message(params$minConf)
message(params$bestLift)
message(params$lenRules)
message(params$ratio)
```

---

arlc\_get\_network\_dataset

*Get Network Dataset*

---

**Description**

This function reads a network dataset from a GML file, assigns node names, and calculates various properties of the graph such as total edges, total nodes, and average degree.

**Usage**

```
arlc_get_network_dataset(file_path, label)
```

**Arguments**

file\_path      The file path to the GML file to be loaded.  
 label          A label for the graph.

**Details**

This function loads a network dataset from a specified GML file and computes basic graph properties.

**Value**

A list containing the graph object and its properties: total edges, total nodes, and average degree.

**Examples**

```
# Create a sample transactions dataset
sample_gml_file <- system.file("extdata", "karate.gml", package = "arlcclustering")
loaded_karate <- arlc_get_network_dataset(sample_gml_file, "Karate Club")
message(loaded_karate$graph)
message(loaded_karate$graphLabel)
message(loaded_karate$totalEdges)
message(loaded_karate$graphEdges)
message(loaded_karate$totalNodes)
message(loaded_karate$graphNodes)
message(loaded_karate$averageDegree)
```

---

arlc\_get\_NonR\_rules      *Get Non-Redundant Rules*

---

**Description**

This function takes a set of gross rules, removes redundant rules, and returns the total number of non-redundant rules along with the non-redundant rules.

**Usage**

```
arlc_get_NonR_rules(gross_rules)
```

**Arguments**

gross\_rules      A vector or dataframe of gross rules.

**Details**

This function cleans the gross rules and provides non-redundant rules.

**Value**

A list containing the total number of non-redundant rules and the non-redundant rules.

**Examples**

```
library(arlclustering)
# Create a sample transactions dataset
sample_gml_file <- system.file("extdata", "karate.gml", package = "arlclustering")
g <- arlc_get_network_dataset(sample_gml_file, "Karate Club")
trans <- arlc_gen_transactions(g$graph)
supportRange <- seq(0.1, 0.2, by = 0.1)
Conf <- 0.5
params <- arlc_get_apriori_thresholds(trans, supportRange, Conf)
grossRules <- arlc_gen_gross_rules(trans, params$minSupp, params$minConf, 1, params$lenRules)
nonRR_rules <- arlc_get_NonR_rules(grossRules$GrossRules)
```

---

arlc\_get\_significant\_rules

*Get Significant Rules*

---

**Description**

This function takes all transactions and a set of non-redundant rules as input, and returns significant rules based on a specified method and adjustment.

**Usage**

```
arlc_get_significant_rules(all_trans, nonRR_rules)
```

**Arguments**

all_trans	A dataframe containing all transactions.
nonRR_rules	A list of non-redundant rules.

**Details**

This function filters significant rules from a set of non-redundant rules.

**Value**

A list containing the total number of significant non-redundant rules and the significant rules themselves.

**Examples**

```
library(arlclustering)
# Create a sample transactions dataset
sample_gml_file <- system.file("extdata", "karate.gml", package = "arlclustering")
g <- arlc_get_network_dataset(sample_gml_file, "Karate Club")
trans <- arlc_gen_transactions(g$graph)
supportRange <- seq(0.1, 0.2, by = 0.1)
Conf <- 0.5
params <- arlc_get_apriori_thresholds(trans, supportRange, Conf)
grossRules <- arlc_gen_gross_rules(trans, params$minSupp, params$minConf, 1, params$lenRules)
nonRR_rules <- arlc_get_NonR_rules(grossRules$GrossRules)
NonRRSig_rules <- arlc_get_significant_rules(trans, nonRR_rules$FiltredRules)
```

---

arlc\_is\_numeric\_vector

*Check if a Vector is Numeric*

---

**Description**

This function checks if all elements of a vector are numeric.

**Usage**

```
arlc_is_numeric_vector(x)
```

**Arguments**

x                    A vector.

**Value**

TRUE if all elements are numeric, FALSE otherwise.

**Examples**

```
arlc_is_numeric_vector(c(1, 2, 3))
arlc_is_numeric_vector(c(1, "a", 3))
```

---

arlc_list_to_df	<i>Convert List of Vectors to Data Frame</i>
-----------------	--

---

**Description**

This function converts a list of named vectors to a data frame.

**Usage**

```
arlc_list_to_df(lst)
```

**Arguments**

lst                    A list of named vectors.

**Value**

A data frame with each element of the list as a row.

**Examples**

```
lst <- list(a = c(x = 1, y = 2), b = c(x = 3, y = 4))
arlc_list_to_df(lst)
```

---

arlc_measure_time	<i>Measure Execution Time of a Function</i>
-------------------	---

---

**Description**

This function measures the execution time of a given function.

**Usage**

```
arlc_measure_time(func, ...)
```

**Arguments**

func                    The function to measure.  
...                    Additional arguments to pass to the function.

**Value**

The result of the function execution.

**Examples**

```
arlc_measure_time(Sys.sleep, 1)
```

---

arlc\_normalize\_vector *Normalize a Numeric Vector*

---

**Description**

This function normalizes a numeric vector to have values between 0 and 1.

**Usage**

```
arlc_normalize_vector(x)
```

**Arguments**

x                    A numeric vector.

**Value**

A normalized numeric vector.

**Examples**

```
arlc_normalize_vector(c(1, 2, 3, 4, 5))
```

---

arlc\_replace\_na            *Replace NA with a Specified Value*

---

**Description**

This function replaces NA values in a vector or data frame with a specified value.

**Usage**

```
arlc_replace_na(x, value)
```

**Arguments**

x                    A vector or data frame.  
value                The value to replace NA with.

**Value**

The vector or data frame with NA values replaced.

**Examples**

```
arlc_replace_na(c(1, NA, 3), 0)  
arlc_replace_na(data.frame(a = c(1, NA, 3), b = c(NA, NA, 3)), 0)
```

# Index

arlc\_calculate\_mode, 2  
arlc\_clean\_final\_rules, 3  
arlc\_clusters\_plot, 4  
arlc\_convert\_date\_format, 5  
arlc\_count\_na, 5  
arlc\_df\_summary, 6  
arlc\_fct\_clean\_transactions, 6  
arlc\_fct\_get\_best\_apriori\_thresholds,  
7  
arlc\_file\_exists\_readable, 8  
arlc\_gen\_gross\_rules, 11  
arlc\_gen\_transactions, 12  
arlc\_generate\_clusters, 9  
arlc\_generate\_date\_sequence, 10  
arlc\_generate\_uid, 10  
arlc\_get\_apriori\_thresholds, 12  
arlc\_get\_network\_dataset, 13  
arlc\_get\_NonR\_rules, 14  
arlc\_get\_significant\_rules, 15  
arlc\_is\_numeric\_vector, 16  
arlc\_list\_to\_df, 17  
arlc\_measure\_time, 17  
arlc\_normalize\_vector, 18  
arlc\_replace\_na, 18