

Package: cooccure (via r-universe)

July 22, 2026

Title Co-Occurrence Network Construction and Manipulation

Version 0.4.0

Description Constructs co-occurrence networks from several types of input data, such as delimited fields, long/bipartite tables, binary matrices, or wide sequences. Returns tidy edge data frames and supports optional scaling, splitting into several networks, thresholding, and subsetting. Provides eight similarity measures, including Jaccard, cosine, and association strength. Supports export to several network and file formats. Network construction and analysis methods follow Saqr, Lopez-Pernas, Conde, and Hernandez-Garcia (2024, <[doi:10.1007/978-3-031-54464-4_15](https://doi.org/10.1007/978-3-031-54464-4_15)>).

License MIT + file LICENSE

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 7.3.3

Imports graphics, Matrix, methods, stats, utils

Suggests igraph, cograph, Nestimate, tidygraph, testthat (>= 3.0.0), knitr, rmarkdown, shiny, DT

Config/testthat/edition 3

VignetteBuilder knitr

Depends R (>= 3.5.0)

URL <https://github.com/mohsaqr/cooccure>

BugReports <https://github.com/mohsaqr/cooccure/issues>

NeedsCompilation no

Author Mohammed Saqr [aut, cre, cph], Sonsoles López-Pernas [aut, cph], Kamila Misiejuk [aut, cph]

Maintainer Mohammed Saqr <saqr@saqr.me>

Repository <https://cran.r-universe.dev>

Date/Publication 2026-07-22 14:30:16 UTC
RemoteUrl <https://github.com/cran/cooccure>
RemoteRef HEAD
RemoteSha 5c954f6a58bc1ad165851e1a2c6a7edcfd59663f

Contents

actor_genres	2
actors	3
as.data.frame.summary.cooccurrence	4
as_cograph	5
as_igraph	5
as_matrix	6
as_netobject	7
as_tidygraph	7
cooccurrence	8
demo	14
launch_app	14
movies	15
plot.cooccurrence	16
print.cooccurrence	16
print.summary.cooccurrence	17
summary.cooccurrence	17
Index	19

actor_genres	<i>IMDB actor-genre long table (1970-2024)</i>
--------------	--

Description

Long-format table mapping each of the 624 actors in [actors](#) to every genre of every movie they appeared in. Use this to build an actor co-occurrence network grouped by genre: which actors share the same genres? Pass field = "actor" and by = "genre" to [cooccurrence](#).

Usage

actor_genres

Format

A data frame with 2,502 rows and 2 variables:

actor Actor name.

genre Genre label (one row per actor-genre combination).

Source

<https://developer.imdb.com/non-commercial-datasets/>

Examples

```
head(actor_genres)
cooccurrence(actor_genres, field = "actor", by = "genre", similarity = "jaccard")
```

actors	<i>IMDB actor-movie long table (1970-2024)</i>
--------	--

Description

Long-format bipartite table linking actors to movies in [movies](#). Pre-filtered to the 624 actors who appear in at least two movies, so all similarity measures compute instantly. Pass `field = "actor"` and `by = "tconst"` to [cooccurrence](#) to build an actor co-appearance network.

Usage

```
actors
```

Format

A data frame with 1,267 rows and 7 variables:

actor Actor name.

tconst IMDB title identifier linking to [movies](#).

primaryTitle Movie title.

startYear Release year (integer).

decade Release decade as a character string.

genres Comma-separated genre labels for the linked movie.

averageRating IMDB average user rating for the linked movie.

Source

<https://developer.imdb.com/non-commercial-datasets/>

Examples

```
head(actors)
cooccurrence(actors, field = "actor", by = "tconst", similarity = "jaccard")
```

as.data.frame.summary.cooccurrence

Tidy node or group table from a cooccurrence summary

Description

Returns the summary's node-level metrics (one row per node) or its group-level metrics (one row per group) as a plain `data.frame`, so callers never have to reach into the summary object's internals.

Usage

```
## S3 method for class 'summary.cooccurrence'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  what = c("nodes", "groups"),
  ...
)
```

Arguments

<code>x</code>	A <code>summary.cooccurrence</code> object.
<code>row.names</code>	Passed to <code>as.data.frame</code> ; unused.
<code>optional</code>	Passed to <code>as.data.frame</code> ; unused.
<code>what</code>	Character. "nodes" (default) for one row per node with degree, strength, count strength, and frequency; "groups" for one row per group when the network was built with <code>split_by</code> .
<code>...</code>	Ignored.

Value

A base `data.frame`. For `what = "nodes"`, one row per node; for `what = "groups"`, one row per group (zero rows when the network was not split).

Examples

```
res <- cooccurrence(list(c("A","B","C"), c("B","C"), c("A","C")))
as.data.frame(summary(res))
```

as_cograph	<i>Convert to cograph network</i>
------------	-----------------------------------

Description

Creates a `cograph_network` object from a cooccurrence edge list, compatible with `cograph::splot()` and other `cograph` functions.

Usage

```
as_cograph(x, ...)

## S3 method for class 'cooccurrence'
as_cograph(x, ...)
```

Arguments

<code>x</code>	A cooccurrence data frame.
<code>...</code>	Ignored.

Value

A `cograph_network` object.

Examples

```
res <- cooccurrence(list(c("A","B","C"), c("B","C"), c("A","C")))
if (requireNamespace("cograph", quietly = TRUE)) {
  net <- as_cograph(res)
  net$n_nodes
}
```

as_igraph	<i>Convert to igraph</i>
-----------	--------------------------

Description

Creates an undirected, weighted `igraph` graph from a cooccurrence edge list.

Usage

```
as_igraph(x, ...)

## S3 method for class 'cooccurrence'
as_igraph(x, ...)
```

Arguments

`x` A cooccurrence data frame.
`...` Passed to `igraph::graph_from_data_frame`.

Value

An `igraph` object.

Examples

```
res <- cooccurrence(list(c("A","B","C"), c("B","C"), c("A","C")))
if (requireNamespace("igraph", quietly = TRUE)) {
  g <- as_igraph(res)
  igraph::vcount(g)
}
```

as_matrix	<i>Extract the co-occurrence matrix</i>
-----------	---

Description

Returns the full square co-occurrence matrix (normalized + scaled). Use `type = "raw"` for the raw count matrix.

Usage

```
as_matrix(x, ...)
```

```
## S3 method for class 'cooccurrence'
as_matrix(x, type = c("normalized", "raw"), ...)
```

Arguments

`x` A cooccurrence data frame.
`...` Ignored.
`type` Character. "normalized" (default) or "raw".

Value

A numeric matrix.

Examples

```
res <- cooccurrence(list(c("A","B","C"), c("B","C"), c("A","C")))
as_matrix(res)
as_matrix(res, type = "raw")
```

as_netobject	<i>Convert to Nestimate netobject</i>
--------------	---------------------------------------

Description

Creates a netobject from a cooccurrence edge list, compatible with `Nestimate::centrality()`, `Nestimate::bootstrap_network()`, etc.

Usage

```
as_netobject(x, ...)

## S3 method for class 'cooccurrence'
as_netobject(x, ...)
```

Arguments

x	A cooccurrence data frame.
...	Ignored.

Value

A netobject with class `c("netobject", "cograph_network")`.

Examples

```
res <- cooccurrence(list(c("A","B","C"), c("B","C"), c("A","C")))
if (requireNamespace("Nestimate", quietly = TRUE)) {
  net <- as_netobject(res)
  net$n_nodes
}
```

as_tidygraph	<i>Convert to tidygraph</i>
--------------	-----------------------------

Description

Creates a `tbl_graph` from a cooccurrence edge list.

Usage

```
as_tidygraph(x, ...)

## S3 method for class 'cooccurrence'
as_tidygraph(x, ...)
```

Arguments

`x` A cooccurrence data frame.
`...` Ignored.

Value

A `tbl_graph` object.

Examples

```
res <- cooccurrence(list(c("A","B","C"), c("B","C"), c("A","C")))
if (requireNamespace("tidygraph", quietly = TRUE) &&
    requireNamespace("igraph", quietly = TRUE)) {
  as_tidygraph(res)
}
```

cooccurrence	<i>Build a co-occurrence network</i>
--------------	--------------------------------------

Description

Constructs an undirected co-occurrence network from various input formats and returns a tidy edge data frame. Argument names follow the citenets convention.

Usage

```
cooccurrence(
  data,
  field = NULL,
  by = NULL,
  sep = NULL,
  weight_by = NULL,
  split_by = NULL,
  similarity = c("none", "jaccard", "cosine", "inclusion", "association", "dice",
    "equivalence", "relative"),
  counting = c("full", "fractional", "attention"),
  scale = NULL,
  threshold = 0,
  min_occur = 1L,
  top_n = NULL,
  output = c("default", "gephi", "igraph", "cograph", "matrix"),
  vars = NULL,
  actor = NULL,
  action = NULL,
  time = NULL,
  session = NULL,
  order = NULL,
```

```

    time_threshold = 900,
    group = NULL,
    aggregate_by = NULL,
    aggregate = c("sum", "mean", "min", "max"),
    window = NULL,
    lambda = 1,
    ...
)

co(
  data,
  field = NULL,
  by = NULL,
  sep = NULL,
  weight_by = NULL,
  split_by = NULL,
  similarity = c("none", "jaccard", "cosine", "inclusion", "association", "dice",
    "equivalence", "relative"),
  counting = c("full", "fractional", "attention"),
  scale = NULL,
  threshold = 0,
  min_occur = 1L,
  top_n = NULL,
  output = c("default", "gephi", "igraph", "cograph", "matrix"),
  vars = NULL,
  actor = NULL,
  action = NULL,
  time = NULL,
  session = NULL,
  order = NULL,
  time_threshold = 900,
  group = NULL,
  aggregate_by = NULL,
  aggregate = c("sum", "mean", "min", "max"),
  window = NULL,
  lambda = 1,
  ...
)

```

Arguments

data	Input data. Accepts: • A data.frame with a delimited column (field + sep). • A data.frame in long/bipartite format (field + by). • A binary (0/1 or TRUE/FALSE) data.frame or matrix (auto-detected). • A one-hot or count table with vars naming the indicator columns; other columns (ids, metadata) are ignored. • A wide sequence data.frame or matrix (non-binary).
------	--

	• A list of character vectors (each element is a transaction). • A <code>nestimate_data</code> object from <code>Nestimate::prepare()</code>; its <code>sequence_data</code> is used, so event logs can be sessionized there and networked here.
field	Character. The entity column — determines what the nodes are. For delimited format, a single column split by <code>sep</code> . For long/bipartite, the item column. For multi-column delimited, a vector of column names pooled per row. Use <code>field = "all"</code> for wide sequence data (e.g. TraMineR / tna format) where every column is a time point and cell values are the items.
by	Character or NULL. Grouping column for long/bipartite format. Each unique value defines one transaction.
sep	Character or NULL. Separator for splitting delimited fields.
weight_by	Character or NULL. Column name containing a numeric association strength for each entity-transaction pair. Only accepted for long format (<code>field + by</code>). When supplied, each entity contributes its weight rather than 1, so $C_{ij} = \sum_d w_{id} \cdot w_{jd}$. Typical use: topic-document probability matrices from LDA or similar models.
split_by	Character or NULL. Column name to split the data by before computing co-occurrence. A separate network is computed per group and the results are combined into a single data frame with an additional group column. Only works with <code>data.frame</code> inputs.
similarity	Character. Similarity measure: "none" Raw co-occurrence counts. "jaccard" $C_{ij} / (f_i + f_j - C_{ij})$. "cosine" Salton's cosine: $C_{ij} / \sqrt{f_i \cdot f_j}$. "inclusion" Simpson coefficient: $C_{ij} / \min(f_i, f_j)$. "association" Association strength: $C_{ij} / (f_i \cdot f_j)$ (van Eck & Waltman, 2009). "dice" $2C_{ij} / (f_i + f_j)$. "equivalence" Salton's cosine squared: $C_{ij}^2 / (f_i \cdot f_j)$. "relative" Row-normalized: each row sums to 1.
counting	Character. Counting method: "full" Each co-occurring pair adds 1 regardless of transaction size. Default. "fractional" Each pair adds $1/(n_i - 1)$ where n_i is the number of items in transaction i. Transactions with many items contribute less per pair (Perianes-Rodriguez et al., 2016). "attention" Each pair within a transaction contributes $\exp(- pos_i - pos_j /\lambda)$ — closer positions give a stronger edge, distant pairs decay. Requires ordered transactions (list / wide / delimited / windowed input). The decay rate λ is controlled by the <code>lambda</code> argument.
scale	Character or NULL. Optional scaling applied to weights after similarity normalization: NULL or "none" No scaling. "minmax" Min-max to $[0, 1]$. "log" Natural log: $\log(1 + w)$. "log10" Log base 10: $\log_{10}(1 + w)$.

	"binary" Binary: 1 if $w > 0$, else 0.
	"zscore" Z-score standardization.
	"sqrt" Square root.
	"proportion" Divide by sum of all weights.
threshold	Numeric. Minimum edge weight to retain, applied after similarity and scaling. The default 0 means no filtering rather than "drop negative weights", so centering scalings such as <code>scale = "zscore"</code> keep their negative half. Pass a positive value to filter.
min_occur	Integer. Minimum entity frequency. Entities appearing in fewer than <code>min_occur</code> transactions are dropped. Default 1.
top_n	Integer or NULL. Keep only the top <code>top_n</code> edges by weight. When <code>split_by</code> is used, applied per group. Default NULL (all edges).
output	Character. Column naming convention for the output: "default" from, to, weight, count. "gephi" Source, Target, Weight, Type (= "Undirected"). Ready for Gephi import. "igraph" Returns an igraph graph object directly. "cograph" Returns a cograph_network object directly. "matrix" Returns the square co-occurrence matrix.
vars	Column specification or NULL. The indicator (one-hot) columns, one column per item. Resolved like select in subset , so all of these work: a bare range (<code>vars = A:D</code>), bare names (<code>vars = c(A, B, C)</code>), positions (<code>vars = 2:5</code>), negative selection (<code>vars = -c(id, note)</code>), a logical mask, or a character vector. Every other column — ids, timestamps, metadata — is ignored, so a one-hot table that also carries identifier columns needs no pre-processing. Cell values are read as presence: any non-zero, non-NA value marks the item present, so count tables such as document-term matrices work as well as 0/1 and TRUE/FALSE. NA counts as absent. Cannot be combined with <code>field</code> , <code>by</code> , or <code>sep</code> . A purely binary table with no extra columns is still auto-detected without <code>vars</code> .
actor	Character vector or NULL. Column(s) identifying who performed the action. When NULL, all rows are treated as one actor. Only used with <code>action</code> .
action	Character or NULL. Column holding the event/state for raw event-log input. When supplied, the log is sessionized into ordered sequences first and each session becomes one transaction, so <code>window</code> and <code>counting = "attention"</code> apply. Requires the Nestimate package, which performs the sessionization.
time	Character or NULL. Timestamp column. Accepts ISO8601, Unix time, and common date/time formats. When NULL, row order defines the sequence. Only used with <code>action</code> .
session	Character vector or NULL. Column(s) giving an explicit session grouping. Combined with <code>time</code> , sessions are further split on time gaps. Only used with <code>action</code> .
order	Character or NULL. Column used to break ties when timestamps are identical. Only used with <code>action</code> .
time_threshold	Numeric. Maximum gap in seconds between consecutive events before a new session starts. Use Inf to keep each actor as a single sequence regardless of gaps. Default 900 (15 minutes). Only used with <code>action</code> and <code>time</code> .

group	Character or NULL. Alias for split_by.
aggregate_by	Character or NULL. Column name to group the data by before computing co-occurrence. For each unique value, the per-group network is computed (with the chosen similarity, counting, scale, window); the per-group edge weights are then combined across groups via aggregate into ONE final network. Differs from split_by, which keeps groups separate. Cannot be combined with split_by. Only applies to data frame inputs.
aggregate	Character. How to combine edge weights across groups when aggregate_by is used: "sum" (default), "mean", "min", or "max". The count column is always summed. threshold and top_n are applied AFTER aggregation.
window	Integer or NULL. Sliding-window size for categorical time-series / ordered-sequence input. When set to an integer $w \geq 2$, every window of w consecutive positions in a sequence becomes a mini-transaction; states inside the same window co-occur. Sequences shorter than w contribute no transactions. Only applies to ordered formats: wide (field = "all") and list. Default NULL (whole sequence treated as one transaction — bag of states).
lambda	Numeric. Decay rate for counting = "attention". Higher lambda $\rightarrow$ slower decay $\rightarrow$ distant pairs still contribute. Default 1.0, matching the tna package. Ignored for other counting methods.
...	Currently unused.

Value

Depends on output:

- "default": A cooccurrence data frame with columns from, to, weight, count (and group when split_by is used).
- "gephi": A data frame with columns Source, Target, Weight, Type, Count. Ready for Gephi CSV import.
- "igraph": An igraph graph object.
- "cograph": A cograph_network object.
- "matrix": A square numeric co-occurrence matrix.

For the data frame outputs, rows are sorted by weight descending and attributes store the full matrix, item frequencies, and parameters.

References

van Eck, N. J., & Waltman, L. (2009). How to normalize co-occurrence data? An analysis of some well-known similarity measures. *Journal of the American Society for Information Science and Technology*, 60(8), 1635–1651.

Examples

```
# Delimited keywords
df <- data.frame(
  id = 1:4,
```

```

    keywords = c("network; graph", "graph; matrix; network",
                  "matrix; algebra", "network; algebra; graph")
  )
  cooccurrence(df, field = "keywords", sep = ";")

# Split by a grouping variable
df$year <- c(2020, 2020, 2021, 2021)
cooccurrence(df, field = "keywords", sep = ";", split_by = "year")
cooccurrence(df, field = "keywords", sep = ";", group = "year")

# List of transactions with Jaccard similarity
cooccurrence(list(c("A","B","C"), c("B","C"), c("A","C")),
              similarity = "jaccard")

# Short alias
co(df, field = "keywords", sep = ";", similarity = "cosine")

# Windowed co-occurrence on a categorical time series. With
# window = 2 only adjacent states co-occur; window = 3 also pairs
# states two positions apart, etc.
seqs <- list(
  c("focus", "focus", "distract", "focus", "confused"),
  c("focus", "distract", "distract", "focus")
)
cooccurrence(seqs, window = 2)

# Weighted long format (e.g. LDA topic-document probabilities)
theta <- data.frame(
  doc   = c("d1","d1","d1","d2","d2","d3","d3"),
  topic = c("T1","T2","T3","T1","T3","T2","T3"),
  prob  = c(0.6, 0.3, 0.1, 0.4, 0.6, 0.5, 0.5)
)
cooccurrence(theta, field = "topic", by = "doc", weight_by = "prob")

# One-hot / indicator table: name the indicator columns, ignore the rest
onehot <- data.frame(
  doc = c("d1", "d2", "d3"),
  A = c(1, 0, 1), B = c(1, 1, 0), C = c(0, 1, 1)
)
cooccurrence(onehot, vars = c("A", "B", "C"))

# Raw event log: sessionized on a 15-minute gap, then windowed
if (requireNamespace("Nestimate", quietly = TRUE)) {
  events <- data.frame(
    student = rep(c("s1", "s2"), each = 3),
    code    = c("read", "write", "read", "test", "write", "read"),
    stamp   = as.POSIXct("2026-01-01 09:00:00") + c(0, 60, 120, 0, 30, 90)
  )
  cooccurrence(events, actor = "student", action = "code", time = "stamp")
}

```

demo

*Demo actor-movie-genre table***Description**

A small hand-crafted dataset of 30 well-known actors across 10 classic films, with each film carrying its full genre list. Designed for quick exploration in the Shiny app. One row per movie-actor-genre triple, so every pairing of field and by yields a network: actors linked by shared films, genres linked by shared films, or actors linked by shared genres.

Usage

demo

Format

A data frame with 89 rows and 3 variables:

movie Movie title.

actor Actor name.

genre Genre label. Films carry every genre they belong to, so a movie contributes several rows per actor.

Examples

```
head(demo)
# Actors linked by the films they share
cooccurrence(demo, field = "actor", by = "movie", similarity = "jaccard")

# Genres linked by the films they share
cooccurrence(demo, field = "genre", by = "movie", similarity = "jaccard")

# Actors linked by the genres they share
cooccurrence(demo, field = "actor", by = "genre")
```

launch_app

*Launch the cooccure Shiny explorer***Description**

Opens an interactive Shiny application for building and exploring co-occurrence networks. Requires the **shiny** and **DT** packages.

Usage

```
launch_app(...)
```

Arguments

... Passed to `runApp` (e.g. `port`, `launch.browser`).

Value

Called for its side effect (launches the app). No return value.

Examples

```
if (interactive()) {
  launch_app()
}
```

movies	<i>IMDB movie metadata (1970-2024)</i>
--------	--

Description

A sample of 1,000 highly-rated IMDB movies (rating ≥ 7.0 , $\geq 1,000$ votes) released between 1970 and 2024. The genres column is comma-delimited and suitable for use as the field argument to `cooccurrence`.

Usage

movies

Format

- A data frame with 1,000 rows and 7 variables:
- tconst** IMDB title identifier (e.g. "tt0068646").
 - primaryTitle** Movie title.
 - startYear** Release year (integer).
 - genres** Comma-separated genre labels (e.g. "Crime,Drama").
 - decade** Release decade as a character string (e.g. "1970s").
 - averageRating** IMDB average user rating.
 - numVotes** Number of IMDB user votes.

Source

<https://developer.imdb.com/non-commercial-datasets/>

Examples

```
head(movies)
cooccurrence(movies, field = "genres", sep = ",", similarity = "jaccard")
```

plot.cooccurrence	<i>Plot a cooccurrence network</i>
-------------------	------------------------------------

Description

Plots the co-occurrence matrix as a heatmap, an **igraph** network, or a base-R degree distribution.

Usage

```
## S3 method for class 'cooccurrence'
plot(x, type = c("heatmap", "network", "degree"), ...)
```

Arguments

x	A cooccurrence data frame.
type	Character. "heatmap" (default), "network" (requires igraph), or "degree" for a base-R degree distribution bar plot.
...	Passed to the plotting function.

Value

Invisibly returns x.

Examples

```
res <- cooccurrence(list(c("A","B","C"), c("B","C"), c("A","C")))
plot(res)
```

print.cooccurrence	<i>Print a cooccurrence edge list</i>
--------------------	---------------------------------------

Description

Prints a compact network diagnostic header followed by an edge preview.

Usage

```
## S3 method for class 'cooccurrence'
print(x, n = 10L, ...)
```

Arguments

x	A cooccurrence data frame.
n	Integer. Number of rows to show. Default 10.
...	Ignored.

Value

Invisibly returns x.

Examples

```
res <- cooccurrence(list(c("A","B","C"), c("B","C"), c("A","C")))
print(res)
```

```
print.summary.cooccurrence
```

Print a cooccurrence summary

Description

Print a cooccurrence summary

Usage

```
## S3 method for class 'summary.cooccurrence'
print(x, n = 5L, ...)
```

Arguments

x	A summary.cooccurrence object.
n	Integer. Number of top nodes/groups to show. Default 5.
...	Ignored.

Value

Invisibly returns x.

```
summary.cooccurrence
```

Summarize a cooccurrence network

Description

Node- and group-level metrics are available as tidy data frames via [as.data.frame](#).

Usage

```
## S3 method for class 'cooccurrence'
summary(object, ...)
```

Arguments

object	A cooccurrence data frame.
...	Ignored.

Details

For a network built with `split_by`, the returned frame stacks one network per group, so a node pair can occur once per group. Pooled simple-graph metrics (`density`, `mean_degree`, `possible_edges`) are undefined in that case and reported as NA; the per-group table carries the correct values.

Value

Invisibly returns a `summary.cooccurrence` object with network, node, and group-level metrics.

Examples

```
res <- cooccurrence(list(c("A","B","C"), c("B","C"), c("A","C")))
summary(res)

# Node-level metrics as a tidy data frame
as.data.frame(summary(res))
```

Index

* datasets

- actor_genres, [2](#)
- actors, [3](#)
- demo, [14](#)
- movies, [15](#)

- actor_genres, [2](#)
- actors, [2](#), [3](#)
- as.data.frame, [4](#), [17](#)
- as.data.frame.summary.cooccurrence, [4](#)
- as_cograph, [5](#)
- as_igraph, [5](#)
- as_matrix, [6](#)
- as_netobject, [7](#)
- as_tidygraph, [7](#)

- co (cooccurrence), [8](#)
- cooccurrence, [2](#), [3](#), [8](#), [15](#)

- demo, [14](#)

- launch_app, [14](#)

- movies, [3](#), [15](#)

- plot.cooccurrence, [16](#)
- print.cooccurrence, [16](#)
- print.summary.cooccurrence, [17](#)

- runApp, [15](#)

- subset, [11](#)
- summary.cooccurrence, [17](#)