Package 'pmap'

January 9, 2018

Title Process Map Visualization

Version 0.3.2

Description A set of functions to produce the process map visualization for process analysis. It can generate the process map from a process event logs recorded in the process, with the ability of pruning the nodes and/or edges to reduce the complexity of the result.

Depends R (>= 3.2)

License MIT + file LICENSE

Encoding UTF-8

LazyData true

URL https://github.com/twang2218/pmap

BugReports https://github.com/twang2218/pmap/issues

Imports dplyr (>= 0.5.0), data.table (>= 1.9.8), DiagrammeR (>= 0.9.0), stringr

RoxygenNote 6.0.1

Suggests covr, roxygen2, testthat

NeedsCompilation no

Author Tao Wang [aut, cre]

Maintainer Tao Wang <twang2218@gmail.com>

Repository CRAN

Date/Publication 2018-01-09 14:27:21 UTC

R topics documented:

`create_pmap` *Create the process map from event log directly*

Description

Create the process map by analyzing the given `eventlog` and extract the nodes by `generate_nodes()` and edges by `generate_edges()`.

Usage

```
create_pmap(eventlog, distinct_customer = FALSE, target_types = NULL)
```

Arguments

`eventlog`	Event log
`distinct_customer`	Whether should count distinct customer only. Default is `FALSE`.
`target_types`	A vector contains the target event types

Details

```
> eventlog <- data.frame(
    timestamp = c(
      as.POSIXct("2017-10-01"),
      as.POSIXct("2017-10-02"),
      as.POSIXct("2017-10-03"),
      as.POSIXct("2017-10-04"),
      as.POSIXct("2017-10-05"),
      as.POSIXct("2017-10-06"),
      as.POSIXct("2017-10-07"),
      as.POSIXct("2017-10-08"),
      as.POSIXct("2017-10-09"),
      as.POSIXct("2017-10-10")
    ),
    customer_id = c("c1", "c1", "c1", "c1", "c1", "c1", "c1", "c1", "c1", "c1"),
    event_name =  c("a",  "b",  "d",  "a",  "c",  "a",  "b",  "c",  "a",  "d"),
   event_type =  c("campaign", "campaign", "sale", "campaign", "sale", "campaign", "campaign", "sale"
    stringsAsFactors = FALSE
  )
> eventlog
    timestamp customer_id event_name event_type
1  2017-10-01          c1          a   campaign
2  2017-10-02          c1          b   campaign
3  2017-10-03          c1          d       sale
4  2017-10-04          c1          a   campaign
5  2017-10-05          c1          c       sale
```

```
6  2017-10-06          c1          a   campaign
7  2017-10-07          c1          b   campaign
8  2017-10-08          c1          c       sale
9  2017-10-09          c1          a   campaign
10 2017-10-10          c1          d       sale
> p <- create_pmap(eventlog, target_types = c("sale"))
> render_pmap(p)
```

Or for more complex event log:

```
> eventlog <- generate_eventlog(
  size_of_eventlog = 10000,
  number_of_customers = 2000,
  event_catalogs = c("campaign", "sale"),
  event_catalogs_size = c(8, 2))
> head(eventlog)
            timestamp   customer_id         event_name event_type
1 2017-01-01 02:40:20 Customer 1204 Event 7 (campaign)   campaign
2 2017-01-01 03:10:31 Customer 1554 Event 5 (campaign)   campaign
3 2017-01-01 04:01:51  Customer 546 Event 4 (campaign)   campaign
4 2017-01-01 05:04:09 Customer 1119     Event 9 (sale)       sale
5 2017-01-01 06:43:11 Customer 1368 Event 2 (campaign)   campaign
6 2017-01-01 07:43:06  Customer 986 Event 8 (campaign)   campaign
> str(eventlog)
'data.frame':   10000 obs. of  4 variables:
 $ timestamp  : POSIXct, format: "2017-01-01 02:40:20" "2017-01-01 03:10:31" ...
 $ customer_id: chr  "Customer 1204" "Customer 1554" "Customer 546" "Customer 1119" ...
 $ event_name : chr  "Event 7 (campaign)" "Event 5 (campaign)" "Event 4 (campaign)" "Event 9 (sale)" ..
 $ event_type : chr  "campaign" "campaign" "campaign" "sale" ...
> p <- create_pmap(eventlog, target_types = c("sale"))
> render_pmap(p)
```

See Also

prune_edges

`create_pmap_graph` *Create the event graph by given nodes and edges.*

Description

Create the process map graph by specify the nodes and edges

Usage

```
create_pmap_graph(nodes, edges, target_types = NULL)
```

Arguments

nodes	Event list, it should be a `data.frame` containing following columns: • `name`: Event name, will be used as label. (`character`) • `type`: The event type (`character`)
edges	Event transform list, it should be a `data.frame` containing following columns: • `from`: the beginning event of the edge. (`character`) • `to`: the ending event of the edge (`character`) • `amount`: How many of customer affected by the given event. (`numeric`)
target_types	A vector contains the target event types

See Also

create_pmap

Examples

```
eventlog <- generate_eventlog()
nodes <- generate_nodes(eventlog)
head(nodes)
#  # A tibble: 6 x 3
#    name              type   amount
#    <chr>             <chr>   <int>
#  1 Event 1 (normal)  normal    105
#  2 Event 10 (target) target     97
#  3 Event 2 (normal)  normal     94
#  4 Event 3 (normal)  normal     94
#  5 Event 4 (normal)  normal    101
#  6 Event 5 (normal)  normal     95
edges <- generate_edges(eventlog)
head(edges)
#  # A tibble: 6 x 3
#    from             to                amount
#    <chr>            <chr>              <int>
#  1 Event 1 (normal) Event 1 (normal)       8
#  2 Event 1 (normal) Event 10 (target)     10
#  3 Event 1 (normal) Event 2 (normal)      12
#  4 Event 1 (normal) Event 3 (normal)       9
#  5 Event 1 (normal) Event 4 (normal)       7
#  6 Event 1 (normal) Event 5 (normal)      10
p <- create_pmap_graph(nodes, edges, target_types = c("target"))
render_pmap(p)
```

`generate_edges` *Generate edges from event logs*

Description

`eventlog` should be a `data.frame` or `data.table`, which contains, at least, following columns:

- `timestamp`: timestamp column which indicates when event happened. (`POSIXct`)
- `customer_id`: customer identifier. (`character`)
- `event_name`: event name. (`character`)
- `event_type`: event type. (`character`)

Usage

```
generate_edges(eventlog, distinct_customer = FALSE, target_types = NULL)
```

Arguments

`eventlog`	Event logs
`distinct_customer`	Whether should only count unique customer
`target_types`	A vector contains the target event types. By default, it's `NULL`, which means every paths count. If it's contains the target event type, then only paths reaches the target event count.

Value

a `data.frame` of edges with `from`, `to` and `amount` columns.

Examples

```
# ------------------------------------------------------
# Generating edges and count every paths no matter whether
# it's from the same customer or not.
# ------------------------------------------------------
eventlog <- generate_eventlog()
edges <- generate_edges(eventlog)
head(edges)
# # A tibble: 6 x 3
#   from             to                amount
#   <chr>            <chr>              <int>
# 1 Event 1 (normal) Event 1 (normal)      13
# 2 Event 1 (normal) Event 10 (target)      3
# 3 Event 1 (normal) Event 2 (normal)       7
# 4 Event 1 (normal) Event 3 (normal)       9
# 5 Event 1 (normal) Event 4 (normal)      11
# 6 Event 1 (normal) Event 5 (normal)      14
str(edges)
# Classes ‘tbl_df’, ‘tbl’ and 'data.frame': 100 obs. of  3 variables:
#  $ from  : chr  "Event 1 (normal)" "Event 1 (normal)" "Event 1 (normal)" "Event 1 (normal)" ...
#  $ to    : chr  "Event 1 (normal)" "Event 10 (target)" "Event 2 (normal)" "Event 3 (normal)" ...
#  $ amount: int  13 3 7 9 11 14 8 12 16 15 ...
#  - attr(*, ".internal.selfref")=<externalptr>
# ------------------------------------------------------
```

```
# Generate edges by specify the target types, and the paths
# not reaching the target type events will be ignored.
# -----------------------------------------------------
edges <- generate_edges(eventlog, target_types = c("target"))
str(edges)
# Classes 'tbl_df', 'tbl' and 'data.frame': 80 obs. of  3 variables:
#  $ from  : chr  "Event 1 (normal)" "Event 1 (normal)" "Event 1 (normal)" "Event 1 (normal)" ...
#  $ to    : chr  "Event 1 (normal)" "Event 10 (target)" "Event 2 (normal)" "Event 3 (normal)" ...
#  $ amount: int  12 3 7 7 11 9 7 8 16 15 ...
#  - attr(*, ".internal.selfref")=<externalptr>
```

`generate_eventlog` *Generate random event log*

Description

This function provides the ability to randomly generate the `eventlog` data frame based on given parameters.

Usage

```
generate_eventlog(
     size_of_eventlog = 1000,
     number_of_customers = 20,
     event_catalogs = c("normal", "target"),
     event_catalogs_size = c(8, 2))
```

Arguments

`size_of_eventlog`	The size of generated event log
`number_of_customers`	How many customers in the simulation
`event_catalogs`	A data frame contains the event catalog
`event_catalogs_size`	How many event types in each event catalog

Examples

```
eventlog <- generate_eventlog(
     size_of_eventlog = 10000,
     number_of_customers = 2000,
     event_catalogs = c("campaign", "sale"),
     event_catalogs_size = c(10, 4)
     )

str(eventlog)
# 'data.frame': 10000 obs. of  4 variables:
```

```
#  $ timestamp  : POSIXct, format: "2017-01-01 02:16:16" ...
#  $ customer_id: chr  "Customer 107" "Customer 1828" "Customer 587" "Customer 1666" ...
#  $ event_name : chr  "Event 4 (campaign)" "Event 11 (sale)" "Event 7 (campaign)" ...
#  $ event_type : chr  "campaign" "sale" "campaign" "sale" ...
head(eventlog)
#             timestamp   customer_id         event_name event_type
# 1 2017-01-01 02:16:16  Customer 107 Event 4 (campaign)   campaign
# 2 2017-01-01 03:04:22 Customer 1828    Event 11 (sale)       sale
# 3 2017-01-01 03:36:35  Customer 587 Event 7 (campaign)   campaign
# 4 2017-01-01 05:00:11 Customer 1666    Event 14 (sale)       sale
# 5 2017-01-01 05:38:24 Customer 1287    Event 11 (sale)       sale
# 6 2017-01-01 05:48:22 Customer 1286 Event 7 (campaign)   campaign
```

`generate_nodes` *Generate nodes from event logs*

Description

`eventlog` should be a `data.frame`, which contains, at least, following columns:

- `event_name`: event name. (`character`)
- `event_type`: event type. (`character`)
- `amount`: how many time this event happened in the `eventlog`

`generate_nodes()` will generate the node list from the given `eventlog` for the graph purpose.

Usage

```
generate_nodes(eventlog, distinct_customer = FALSE)
```

Arguments

`eventlog`	Event logs
`distinct_customer`	Whether should only count unique customer

Value

a nodes `data.frame` which represents a event list, it contains `name`, `type` and `amount` columns.

Examples

```
# ------------------------------------------------------
# Generate nodes from eventlog and count every event
# ------------------------------------------------------
eventlog <- generate_eventlog(10000, 100)
nodes <- generate_nodes(eventlog)
print(nodes)
# # A tibble: 10 x 3
```

```
#    name              type   amount
#    <chr>             <chr>   <int>
#  1 Event 1 (normal)  normal    958
#  2 Event 10 (target) target    948
#  3 Event 2 (normal)  normal   1011
#  4 Event 3 (normal)  normal   1030
#  5 Event 4 (normal)  normal   1072
#  6 Event 5 (normal)  normal    968
#  7 Event 6 (normal)  normal   1020
#  8 Event 7 (normal)  normal    978
#  9 Event 8 (normal)  normal   1003
# 10 Event 9 (target)  target   1012
#
# ------------------------------------------------------
# Generate nodes and only count by unique customer.
# ------------------------------------------------------
#
nodes <- generate_nodes(eventlog, distinct_customer = TRUE)
nodes
# # A tibble: 10 x 3
#    name              type   amount
#    <chr>             <chr>   <int>
#  1 Event 1 (normal)  normal    100
#  2 Event 10 (target) target    100
#  3 Event 2 (normal)  normal    100
#  4 Event 3 (normal)  normal    100
#  5 Event 4 (normal)  normal    100
#  6 Event 5 (normal)  normal    100
#  7 Event 6 (normal)  normal    100
#  8 Event 7 (normal)  normal    100
#  9 Event 8 (normal)  normal    100
# 10 Event 9 (target)  target    100
```

`get_attrs_desc` *Get an attribute key-pair string from an object*

Description

Get an attribute list string from an object

Usage

```
get_attrs_desc(object)
```

Arguments

`object`	Given object, can be `list`, `matrix`, or `data.frame`

Examples

```
print(df)
#   id   name is_manager
# 1  1   Jane      FALSE
# 2  2   John      FALSE
# 3  3   Eric      FALSE
# 4  4 Selena       TRUE
get_attrs_desc(df)
# [1] "id: 1\nname: Jane\nis_manager: FALSE"
# [2] "id: 2\nname: John\nis_manager: FALSE"
# [3] "id: 3\nname: Eric\nis_manager: FALSE"
# [4] "id: 4\nname: Selena\nis_manager: TRUE"
```

`prune_edges` *Prune edges based on given percentage*

Description

Prune edges based on given percentage

Usage

```
prune_edges(p, percentage = 0.2)
```

Arguments

`p`	process map object created by `create_pmap_graph()` function
`percentage`	how many percentage of the edges should be pruned.

Details

Create an event log

```
> library(dplyr)
> library(pmap)
> eventlog <- generate_eventlog(
  size_of_eventlog = 10000,
  number_of_customers = 2000,
  event_catalogs = c("campaign", "sale"),
  event_catalogs_size = c(10, 4))
> head(eventlog)
            timestamp   customer_id          event_name event_type
1 2017-01-01 02:14:50  Customer 345  Event 1 (campaign)   campaign
2 2017-01-01 02:26:24 Customer 1625  Event 2 (campaign)   campaign
3 2017-01-01 03:48:12 Customer 1901     Event 12 (sale)       sale
4 2017-01-01 03:57:54 Customer 1029 Event 10 (campaign)   campaign
5 2017-01-01 07:46:54  Customer 215 Event 10 (campaign)   campaign
```

```
6 2017-01-01 09:44:51 Customer 1354  Event 1 (campaign)   campaign
> str(eventlog)
'data.frame': 10000 obs. of  4 variables:
 $ timestamp  : POSIXct, format: "2017-01-01 02:14:50" "2017-01-01 02:26:24" ...
 $ customer_id: chr  "Customer 345" "Customer 1625" "Customer 1901" "Customer 1029" ...
 $ event_name : chr  "Event 1 (campaign)" "Event 2 (campaign)" "Event 12 (sale)" "Event 10 (campaign)"
 $ event_type : chr  "campaign" "campaign" "sale" "campaign" ...
```

Create a process map from the event log and render it directly.

```
> p <- create_pmap(eventlog, target_types = c("sale"))
> render_pmap(p)
```

As you can see the event map is very messy. Let's apply the `prune_edges()` to remove 50 percent edges.

```
> p %>% prune_edges(0.5) %>% render_pmap()
```

It's cleaner, we can clean it further by remove 30 percent nodes with `prune_nodes()` function.

```
> p %>% prune_edges(0.5) %>% prune_nodes(0.3) %>% render_pmap()
```

`prune_nodes` *Prune nodes based on given percentage*

Description

Prune nodes based on given percentage

Usage

```
prune_nodes(p, percentage = 0.2, rank = "amount")
```

Arguments

`p`	process map object created by `create_pmap_graph()` function
`percentage`	how many percentage of the nodes should be pruned.
`rank`	how to rank the nodes. `amount` means ranking the nodes by `amount` column (default); `in_degree` means ranking the nodes by `in_degree`; `out_degree` means ranking the nodes by `out_degree`;

See Also

prune_edges

Examples

```
library(dplyr)
p <- generate_eventlog() %>% create_pmap(target_types = c("target"))
DiagrammeR::node_count(p)
# [1] 10
p <- prune_nodes(p, percentage = 0.5)
DiagrammeR::node_count(p)
# [1] 5
```

`render_pmap` *Render the process map*

Description

Basically, this function just called `DiagrammeR::render_graph()`

Usage

```
render_pmap(p, title = NULL)
```

Arguments

`p`	the process map object created by `create_pmap_graph()` function
`title`	The title of rendered graph

See Also

create_pmap

Examples

```
library(dplyr)
p <- generate_eventlog() %>% create_pmap(target_types = c("target"))
render_pmap
```

Index