Package ‘gstsm’

November 8, 2021

Title Generalized Spatial-Time Sequence Miner

Version 0.0.1

Description Implementations of the algorithms present in the future article Generalized Discovery of Tight Space-Time Sequences, original title (Castro Filho, A. J. ; Borges, H. ; Pacitti, Esther ; Porto, F. ; Coutinho, R. ; Ogasawara, E. . Generalização de Mineração de Sequências Restritas no Espaço e no Tempo. In: XXXVI SBBD - Simpósio Brasileiro de Banco de Dados, 2021).

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.1.2

Imports digest

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Cássio Souza [aut, cre],
Jorge Rodrigues [aut],
Eduardo Ogasawara [ctb] (<https://orcid.org/0000-0002-0466-0626>),
Antonio Filho [ctb],
CEFET/RJ [cph]

Maintainer Cássio Souza <cassiofb.souza@gmail.com>

Repository CRAN

Date/Publication 2021-11-08 15:30:02 UTC

R topics documented:

`find_kernel_ranged_group` *Algorithm 3: Find Kernel Ranged Group*

Description

The goal of Algorithm 3 is to find the KRG information for a candidate c It receives as input a candidate c, the set of transactions d from a sliding window of SW, and the thresholds defined by the user (gamma, beta and sigma).

Usage

```
find_kernel_ranged_group(c, d, gamma, beta, adjacency_matrix)
```

Arguments

`c`	candidate
`d`	set of transactions
`gamma`	minimum temporal frequency
`beta`	minimum group size
`adjacency_matrix`	adjacency matrix

Value

Kernel Ranged-Group(s) of c updated

`generate_adjacency_matrix` *Generate Adjacency Matrix*

Description

Helper function that generates an adjacency matrix

Usage

```
generate_adjacency_matrix(spatial_positions, sigma)
```

Arguments

`spatial_positions`	set of spatial positions
`sigma`	max distance between group points

Value

Adjacency Matrix

`generate_candidates` *Algorithm 6: Generate Candidates*

Description

The algorithm combines SRGs that have sequences of size k, received as input, to generate candidates with sequences of size k + 1. Let x and y be SRGs, the conditions for this to occur are (line 3): that we have an intersection of candidates over the time range, intersection over the set of spatial positions (x.g n y.g), and a common subsequence: <x.s2, . . . , x.sk>=<y.s1, . . . , y.sk-1>.

Usage

```
generate_candidates(srg, k, beta)
```

Arguments

`srg`	Solid-Ranged-Groups
`k`	sequence size
`beta`	minimum group size

Value

Ck+1 set of candidates having length k + 1

`gstsm` *Algorithm 1: G-STSM*

Description

This section presents the G-STSM. Our algorithm is designed to the identification of frequent sequences in STS datasets from the concept of SRG. The notion of ranged-group (RG, KRG, and SRG) introduced in the previous section enables for extracting SRG efficiently. The G-STSM is based on the candidate-generating principle. Our goal is to start finding SRGs for sequences of size one. Then we explore the support and the number of occurrences of SRGs for larger sequences with a limited number of scans over the database. To this end, we need to find the range and the set of positions (i. e., the SRG) in which a candidate sequence is frequent in only one scan.

Usage

```
gstsm(sts_dataset, spatial_positions, gamma, beta, sigma)
```

Arguments

`sts_dataset`	STS dataset
`spatial_positions`	set of spatial positions
`gamma`	minimum temporal frequency
`beta`	minimum group size
`sigma`	max distance between group points

Value

Solid-Raged-Groups.

Examples

```
library("gstsm")
events_data_path <-
  system.file("extdata", "made_bangu_6x30.txt", package = "gstsm")

space_time_data_path <-
  system.file("extdata", "positions_2D_30.txt", package = "gstsm")

d <- read.table(
  events_data_path,
  header = FALSE,
  sep = " ",
  dec = ".",
  as.is = TRUE,
  stringsAsFactors = FALSE
)

p <- read.table(
  space_time_data_path,
  header = TRUE,
  sep = " ",
  dec = ".",
  as.is = TRUE,
  stringsAsFactors = FALSE
)

gamma <- 0.8
beta <- 2
sigma <- 1

result <- gstsm::gstsm(d, p, gamma, beta, sigma)
```

merge_kernel_ranged_groups

Algorithm 5: Merge Kernel Ranged Groups

Description

The goal of the Algorithm 5 is to merge KRGs. Let q and u be two different KRGs from the same candidate sequence. They can be merged into a group qu = q U u as long as they have an intersection and qu has a frequency greater than or equal to the minimum frequency defined by the user.

Usage

```
merge_kernel_ranged_groups(c, gamma)
```

Arguments

`c`	candidate
`gamma`	minimum temporal frequency

Value

KRG

merge_open_kernel_ranged_groups

Algorithm 7: Merge Kernel Ranged Groups

Description

The goal of the Algorithm 7 is to stretch KRGs of the same candidate sequence. Its possible if two KRGs have intersection in space and the resulting KRG keeps its frequency equal to or greater than beta.

Usage

```
merge_open_kernel_ranged_groups(c, timestamp, gamma, beta, adjacency_matrix)
```

Arguments

`c`	candidate.
`timestamp`	current timestamp
`gamma`	minimum temporal frequency
`beta`	minimum group size
`adjacency_matrix`	adjacency matrix

Value

Set of updated KRGs

`split_groups` *Split Groups*

Description

Helper function that split groups

Usage

```
split_groups(pos, adjacency_matrix)
```

Arguments

`pos`	sequence occurrence index
`adjacency_matrix`	possible connection between positions

Value

new set based on candidate c found in d.

`validate_and_close` *Algorithm 2: Validate and Close*

Description

The function is shown in Algorithm 2. It receives as input the set of RGs (RG) from a candidate and the minimum size of a group (beta). It starts defining a set of elements that will be removed from the set of RGs (line 2), if it does not have the minimum group size.

Usage

```
validate_and_close(c, gamma, beta)
```

Arguments

`c`	candidate
`gamma`	minimum temporal frequency
`beta`	minimum group size

Value

validated Greedy-Ranged-Groups.

`validate_kernel_ranged_groups` *Algorithm 4: Validate Kernel Ranged Groups*

Description

Its objective is to verify that the user thresholds were observed in each RGs, checking if they can still be stretched by keeping the frequency greater than or equal to the minimum gamma and if the minimum group size beta occurs. It takes as input a set of RGs RG of a candidate sequence, the timestamp of the start of the current sliding window timestamp, the user-defined thresholds gamma and beta.

Usage

```
validate_kernel_ranged_groups(c, timestamp, gamma, beta)
```

Arguments

`c`	candidate
`timestamp`	current timestamp
`gamma`	minimum temporal frequency
`beta`	minimum group size

Value

Validated Kernel-Ranged-Groups.

Index