Package 'opusminer'

February 4, 2020

Type Package

Title OPUS Miner Algorithm for Filtered Top-k Association Discovery

Version 0.1-1

Description Provides a simple R interface to the OPUS Miner algorithm (implemented in C++) for finding the top-k productive, non-redundant itemsets from transaction data. The OPUS Miner algorithm uses the OPUS search algorithm to efficiently discover the key associations in transaction data, in the form of self-sufficient itemsets, using either leverage or lift. See <http://i.giwebb.com/index.php/research/association-discovery/> for more information in relation to the OPUS Miner algorithm.

License GPL-3

Copyright OPUS Miner algorithm and C++ implementation Copyright (C) 2012-2017 Geoffrey I Webb, R interface Copyright (C) 2017 Angus Dempster.

Encoding UTF-8

LazyData true

Imports Rcpp (>= 0.12.9), arules (>= 1.5-0), Matrix (>= 1.2-7), methods

LinkingTo Rcpp

RoxygenNote 6.0.0

NeedsCompilation yes

Author Geoffrey I Webb [aut, cph] (OPUS Miner algorithm and C++ implementation, http://i.giwebb.com/index.php/research/association-discovery/), Christoph Bergmeir [ctb, cre], Angus Dempster [ctb, cph] (R interface)

Maintainer Christoph Bergmeir <christoph.bergmeir@monash.edu>

Repository CRAN

Date/Publication 2020-02-03 23:10:02 UTC

R topics documented:

`opusminer-package` *Filtered Top-k Association Discovery of Self-Sufficient Itemsets*

Description

The `opusminer` package provides an R interface to the OPUS Miner algorithm (implemented in C++), developed by Professor Geoffrey I Webb, for finding the top *k*, non-redundant itemsets on the measure of interest.

Details

OPUS Miner is a branch-and-bound algorithm for efficient discovery of self-sufficient itemsets. For a user-specified *k* and interest measure, OPUS Miner finds the top *k* productive non-redundant itemsets with respect to the specified measure. It is then straightforward to filter out those that are not independently productive with respect to that set, resulting in a set of self-sufficient itemsets.

OPUS Miner is based on the OPUS search algorithm. OPUS is a set enumeration algorithm distinguished by a computationally efficient pruning mechanism that ensures that whenever an item is pruned, it is removed from the entire search space below the parent node.

OPUS Miner systematically traverses viable regions of the search space (using depth-first search), maintaining a collection of the top *k* productive non-redundant itemsets in the search space explored. When all of the viable regions have been explored, the top *k* productive non-redundant itemsets in the search space explored must be the top *k* for the entire search space.

A comprehensive explanation of the algorithm is provided in the article cited below.

References

Webb, G. I., & Vreeken, J. (2014). Efficient Discovery of the Most Interesting Associations. *ACM Transactions on Knowledge Discovery from Data*, 8(3), 1-15. doi: http://dx.doi.org/10.1145/2601433

opus *Filtered Top-k Association Discovery of Self-Sufficient Itemsets*

Description

opus finds the top *k* productive, non-redundant itemsets on the measure of interest (leverage or lift) using the OPUS Miner algorithm.

Usage

```
opus(transactions, k = 100, format = "data.frame", sep = " ",
  print_closures = FALSE, filter_itemsets = TRUE, search_by_lift = FALSE,
  correct_for_mult_compare = TRUE, redundancy_tests = TRUE)
```

Arguments

transactions	A filename, list, or object of class transactions (arules).
k	The number of itemsets to return, an integer (default 100).
format	The output format ("data.frame", default, or "itemsets").
sep	The separator between items (for files, default " ").
print_closures	return the closure for each itemset (default FALSE)
filter_itemsets	filter itemsets that are not independently productive (default TRUE)
search_by_lift	make lift (rather than leverage) the measure of interest (default FALSE)
correct_for_mult_compare	correct alpha for the size of the search space (default TRUE)
redundancy_tests	exclude redundant itemsets (default TRUE)

Details

opus provides an interface to the OPUS Miner algorithm (implemented in C++) to find the top *k* productive, non-redundant itemsets by leverage (default) or lift.

transactions should be a filename, list (of transactions, each list element being a vector of character values representing item labels), or an object of class transactions (arules).

Files should be in the format of a list of transactions, one line per transaction, each transaction (ie, line) being a sequence of item labels, separated by the character specified by the parameter sep (default " "). See, for example, the files at http://fimi.ua.ac.be/data/. (Alternatively, files can be read seaparately using the read_transactions function.)

format should be specified as either "data.frame" (the default) or "itemsets", and any other value will return a list.

Value

The top *k* productive, non-redundant itemsets, with relevant statistics, in the form of a data frame, object of class `itemsets` (`arules`), or a list.

References

Webb, G. I., & Vreeken, J. (2014). Efficient Discovery of the Most Interesting Associations. *ACM Transactions on Knowledge Discovery from Data*, 8(3), 1-15. doi: http://dx.doi.org/10.1145/2601433

Examples

```
## Not run:

result <- opus("mushroom.dat")
result <- opus("mushroom.dat", k = 50)

result[result$self_sufficient, ]
result[order(result$count, decreasing = TRUE), ]

trans <- read_transactions("mushroom.dat", format = "transactions")

result <- opus(trans, print_closures = TRUE)
result <- opus(trans, format = "itemsets")

## End(Not run)
```

`read_transactions` *Read Transaction Data from a File (Fast)*

Description

`read_transactions` reads transaction data from a file fast, providing a significant speed increase over alternative methods for larger files.

Usage

```
read_transactions(filename, sep = " ", format = "list")
```

Arguments

`filename`	A filename.
`sep`	The separator between items (default " ").
`format`	The output format ("list" or "transactions").

Details

read_transactions uses (internally) the readChar function to read transaction data from a file fast. This is substantially faster for larger files than alternative methods.

Files should be in the format of a list of transactions, one line per transaction, each transaction (ie, line) being a sequence of item labels, separated by the character specified by the parameter sep (default " "). See, for example, the files at http://fimi.ua.ac.be/data/.

Value

The transaction data, in the form of a list (of transactions, each list element being a vector of character values representing item labels), or an object of class transactions (arules).

Examples

```
## Not run:

trans <- read_transactions("mushroom.dat")
trans <- read_transactions("mushroom.dat", format = "transactions")

## End(Not run)
```

Index