Package ‘crandep’

June 3, 2022

Title Network Analysis of Dependencies of CRAN Packages

Version 0.3.1

Description The dependencies of CRAN packages can be analysed in a network fashion. For each package we can obtain the packages that it depends, imports, suggests, etc. By iterating this procedure over a number of packages, we can build, visualise, and analyse the dependency network, enabling us to have a bird’s-eye view of the CRAN ecosystem. One aspect of interest is the number of reverse dependencies of the packages, or equivalently the in-degree distribution of the dependency network. This can be fitted by the power law and/or an extreme value mixture distribution <arXiv:2008.03073>, of which functions are provided.

Depends R (>= 3.4)
License GPL (>= 2)

URL https://github.com/clement-lee/crandep

BugReports https://github.com/clement-lee/crandep/issues

Encoding UTF-8
LazyData true

Imports xml2, rvest, stringr, dplyr, igraph, Rcpp
Suggests ggplot2, tibble, visNetwork, knitr, rmarkdown

RoxygenNote 7.1.0

NeedsCompilation yes

SystemRequirements pandoc (>= 1.12.3) - http://pandoc.org

Author Clement Lee [aut, cre] (<https://orcid.org/0000-0003-1785-8671>)
Maintainer Clement Lee <clement.lee.tm@outlook.com>

VignetteBuilder knitr

LinkingTo Rcpp, RcppArmadillo, RcppGSL

Repository CRAN

Date/Publication 2022-06-03 08:30:10 UTC
R topics documented:

- chi_citations                  2
- cran_dependencies             3
- df_to_graph                   3
- dmix                          4
- dupp                          5
- get_dep_all_packages          6
- get_dep_df                    7
- get_graph_all_packages        8
- mcmc_mix                      9
- mcmc_upp                      10
- Smix                          12
- Supp                          13
- topo_sort_kahn                14

Description

A dataset containing the citations of conference papers of the ACM Conference on Human Factors in Computing Systems (CHI) from 1981 to 2019, obtained from the ACM digital library. The resulting citation network can be compared to the dependencies network of CRAN packages, in terms of network-related characteristics, such as degree distribution and sparsity.

Usage

chi_citations

Format

A data frame with 21951 rows and 4 variables:

- from  the unique identifier (in the digital library) of the paper that cites other papers
- to    the unique identifier of the paper that is being cited
- year_from the publication year of the citing paper
- year_to  the publication year of the cited paper

Source

https://dl.acm.org/conference/chi

See Also

cran_dependencies
cran_dependencies

Dependencies of CRAN packages

Description

A dataset containing the dependencies of various types (Imports, Depends, Suggests, LinkingTo, and their reverse counterparts) of more than 14600 packages available on CRAN as of 2020-05-09.

Usage

cran_dependencies

Format

A data frame with 211408 rows and 4 variables:

from  the name of the package that introduced the dependencies
to the name of the package that the dependency is directed towards
type the type of dependency, which can take the follow values (all in lowercase): "depends", "imports", "linking to", "suggests"
reverse a boolean representing whether the dependency is a reverse one (TRUE) or a forward one (FALSE)

Source

The CRAN pages of all the packages available on https://cran.r-project.org

See Also

chi_citations

df_to_graph

Construct the giant component of the network from two data frames

Description

Construct the giant component of the network from two data frames

Usage

df_to_graph(edgelist, nodelist = NULL, gc = TRUE)
Arguments

- **edgelist**: A data frame with (at least) two columns: from and to
- **nodelist**: NULL, or a data frame with (at least) one column: name, that contains the nodes to include
- **gc**: Boolean, if 'TRUE' (default) then the giant component is extracted, if 'FALSE' then the whole graph is returned

Value

An igraph object & a connected graph if gc is 'TRUE'

Examples

```r
from <- c("1", "2", "4")
from <- c("2", "3", "5")
edges <- data.frame(from = from, to = to, stringsAsFactors = FALSE)
nodes <- data.frame(name = c("1", "2", "3", "4", "5"), stringsAsFactors = FALSE)
df_to_graph(edges, nodes)
```

---

**dmix**

*Probability mass function (PMF) of discrete extreme value mixture distribution*

**Description**

*dmix* returns the PMF at x for the discrete extreme value mixture distribution.

**Usage**

```r
dmix(x, u, xi1, xi2, sig, geo, phi, log = FALSE)
```

**Arguments**

- **x**: Vector of positive integers
- **u**: Scalar, positive integer threshold
- **xi1**: Scalar, shape parameter for values below or equal to u
- **xi2**: Scalar, shape parameter of integer generalised Pareto distribution (IGPD), for values above u
- **sig**: Scalar, scale parameter of IGPD, for values above u
- **geo**: Boolean. If 'TRUE', the geometric distribution is used for the values below u. If 'FALSE', the discrete power law is used.
- **phi**: Scalar, exceedance probability of u, between 0.0 and 1.0 exclusive
- **log**: Boolean (default 'FALSE'), whether the PMF should be returned on the log scale.
**Value**

A numeric vector of the same length as x

**See Also**

`Smix` for the corresponding survival function, `dupp` for the probability mass function of the discrete power law.

**Examples**

```r
dmix(10:15, 12, 2.0, 0.5, 1.0, TRUE, 0.2)
dmix(10:15, 12, 2.0, 0.5, 1.0, FALSE, 0.2)
dmix(10:15, 12, 2.0, 0.5, 1.0, FALSE, 0.2, TRUE)
```

---

**dupp**

**Probability mass function (PMF) of discrete power law**

**Description**

`dupp` returns the PMF at x for the discrete power law with exponent \((1.0 / \xi_1 + 1.0)\), for values greater than or equal to u.

**Usage**

```r
dupp(x, u, xi1, log = FALSE)
```

**Arguments**

- `x`: Vector of positive integers
- `u`: Scalar, non-negative integer threshold
- `xi1`: Scalar, a positive real number representing the shape parameter
- `log`: Boolean (default 'FALSE'), whether the PMF should be returned on the log scale.

**Details**

The PMF is proportional to \(x^{(-\alpha)}\), where \(\alpha = 1.0 / \xi_1 + 1.0\). To be a proper PMF, it is normalised by \(1 / hzet(a(x, u))\), where hzeta is the Hurwitz zeta function i.e. \(hzet(y, z) = z^{(-y)} + (z+1)^{(-y)} + (z+2)^{(-y)} + \ldots\). Any values below u will have PMF equal to 0.0. That xi1 is used instead of alpha is for alignment with the parametrisation in `dmix`, `Smix` and `mcmc_mix`.

**Value**

A numeric vector of the same length as x


### Description

`get_dep_all_packages` returns the data frame of dependencies of all packages currently available on CRAN.

### Usage

```r
get_dep_all_packages()
```

### Details

Unlike `get_dep`, there is no boolean argument ‘scrape’, as it is much faster to obtain the dependencies of all packages via ‘tools::CRAN_package_db()’.

### Value

A data frame of dependencies of all CRAN packages

### See Also

- `get_dep` for multiple types of dependencies, and `get_graph_all_packages` for obtaining directly a network of dependencies as an igraph object

### Examples

```r
## Not run:
df.cran <- get_dep_all_packages()
## End(Not run)
```
get_dep_df

## Multiple types of dependencies

### Description

`get_dep` returns a data frame of multiple types of dependencies of a package

### Usage

```r
get_dep_df(name, type, scrape = TRUE)
get_dep_all(name, type, scrape = TRUE)
get_dep(name, type, scrape = TRUE)
```

### Arguments

- **name**: String, name of the package
- **type**: A character vector that contains one or more of the following dependency words: "Depends", "Imports", "LinkingTo", "Suggests", "Enhances", "Reverse depends", "Reverse imports", "Reverse linking to", "Reverse suggests", "Reverse enhances", up to letter case and space replaced by underscore. Alternatively, if 'type = "all"', all ten dependencies will be obtained.
- **scrape**: Boolean. If 'TRUE' (default), the page of the package will be scraped. If 'FALSE', `tools::CRAN_package_db()` will be used. Whether the argument equals 'TRUE' or 'FALSE' should not affect the output, but only the time taken. Usually, the former is faster than the latter for a single package.

### Value

A data frame of dependencies

### See Also

- `get_dep_all_packages` for the dependencies of all CRAN packages, and `get_graph_all_packages` for obtaining directly a network of dependencies as an igraph object

### Examples

```r
get_dep("dplyr", c("Imports", "Depends"))
get_dep("MASS", c("Suggests", "Depends", "Imports"), TRUE) # FALSE will give same result
```
get_graph_all_packages

Graph of dependencies of all CRAN packages

Description

get_graph_all_packages returns an igraph object representing the network of one or more types of dependencies of all CRAN packages.

Usage

get_graph_all_packages(type, gc = TRUE)

Arguments

type A character vector that contains one or more of the following dependency words: "Depends", "Imports", "LinkingTo", "Suggests", "Enhances", "Reverse depends", "Reverse imports", "Reverse linking to", "Reverse suggests", "Reverse enhances", up to letter case and space replaced by underscore. Alternatively, if 'types = "all"', all ten dependencies will be obtained.

gc Boolean, if 'TRUE' (default) then the giant component is extracted, if 'FALSE' then the whole graph is returned

Value

An igraph object & a connected graph if gc is 'TRUE'

See Also

get_dep_all_packages for the dependencies of all CRAN packages in a data frame, and df_to_graph for constructing the giant component of the network from two data frames

Examples

## Not run:
g0.cran.depends <- get_graph_all_packages("depends")
g1.cran.imports <- get_graph_all_packages("reverse imports")

## End(Not run)
**mcmc_mix**

Markov chain Monte Carlo for discrete extreme value mixture distribution

---

### Description

`mcmc_mix` returns the samples from the joint posterior of the parameters \((u, \xi_1, \xi_2, \sigma)\), for fitting the discrete extreme value mixture distribution (DEVMD) to the data \(x\). The samples are obtained using Markov chain Monte Carlo (MCMC).

### Usage

```r
mcmc_mix(
  x,
  u,
  xi1,
  xi2,
  sig,
  geo,
  cont,
  a_phi,
  b_phi,
  a_xi1,
  b_xi1,
  m_xi2,
  s_xi2,
  a_sig,
  b_sig,
  pcont,
  N = 20000L,
  thin = 100L,
  burnin = 20000L,
  print_freq = 10000L
)
```

### Arguments

- **x**: Vector of positive integers, representing the data
- **u**: Scalar, initial value of the positive integer threshold
- **xi1**: Scalar, initial value of the parameter for values below or equal to \(u\)
- **xi2**: Scalar, initial value of the shape parameter of the integer generalised Pareto distribution (IGPD), for values above \(u\)
- **sig**: Scalar, initial value of the scale parameter of IGPD, for values above \(u\)
- **geo**: Boolean. If 'TRUE', the geometric distribution is used for the values below \(u\). If 'FALSE', the discrete power law is used.
## mcmc_upp

- **cont**: Boolean, whether the continuity constraint is imposed at $u$
- **a_phi, b_phi, a_xi1, b_xi1, m_xi2, s_xi2, a_sig, b_sig**: Scalars, representing the hyperparameters of the prior distributions of the respective parameters. See details for the specification of the priors.
- **pcont**: Scalar, between 0.0 and 1.0, representing the prior probability of the continuity constrained version, for model selection.
- **N**: Scalar, positive integer representing the length of the output chain, i.e., the number of rows in the returned data frame.
- **thin**: Scalar, positive integer representing the thinning in the MCMC.
- **burnin**: Scalar, non-negative integer representing the burn-in of the MCMC.
- **print_freq**: Scalar, positive integer representing the frequency of printing the sampled values.

### Details

In the MCMC, a componentwise Metropolis-Hastings algorithm is used. Unlike `mcmc_upp`, the threshold $u$ is treated as a parameter in `mcmc_mix` and therefore inferred. The 8 hyperparameters are used in the following priors: $u$ is such that the implied exceedance probability $\phi \sim \text{Uniform}(a_{\phi}, b_{\phi}); x_{11} \sim \text{Uniform}(a_{x11}, b_{x11}); x_{12} \sim \text{Normal}(\text{mean} = m_{x12}, \text{sd} = s_{x12}); \sigma \sim \text{Gamma}(\text{shape} = a_{\sigma}, \text{rate} = b_{\sigma})$. If $pcont = 0.0$, only the unconstrained version of the DEVMD is fitted; if $pcont = 1.0$, only the continuity constrained version is fitted. Setting $pcont$ between 0.0 and 1.0 allows both versions to be fitted, if model selection between the two is of interest.

### Value

A data frame containing $N$ rows and 7 columns which represent (in this order) the 4 parameters ($u$, $x_{11}$, $x_{12}$, $\sigma$), the implied exceedance probability ($\phi$), the log-posterior density ($lpost$), and whether the continuity constraint is imposed (cont).

### See Also

- `mcmc_upp` for MCMC for the discrete power law.

---

**mcmc_upp**

Markov chain Monte Carlo for discrete power law

**Description**

`mcmc_upp` returns the samples from the posterior of $x_{11}$, for fitting the discrete power law to the data $x$. The samples are obtained using Markov chain Monte Carlo (MCMC).
Usage

```r
mcmc_upp(
  x,
  u,
  xi1,
  a_xi1,
  b_xi1,
  N = 20000L,
  thin = 10L,
  burnin = 20000L,
  print_freq = 10000L
)
```

Arguments

- `x` Vector of positive integers, representing the data
- `u` Scalar, non-negative integer threshold
- `xi1` Scalar, initial value of the shape parameter
- `a_xi1` Scalar, lower bound of the uniform distribution as the prior of `xi1`
- `b_xi1` Scalar, upper bound of the uniform distribution as the prior of `xi1`
- `N` Scalar, positive integer representing the length of the output chain i.e. the number of rows in the returned data frame
- `thin` Scalar, positive integer representing the thinning in the MCMC
- `burnin` Scalar, non-negative integer representing the burn-in of the MCMC
- `print_freq` Scalar, positive integer representing the frequency of printing the sampled values

Details

In the MCMC, a componentwise Metropolis-Hastings algorithm is used. Unlike `mcmc_mix`, the threshold `u` is treated as fixed in `mcmc_upp`.

Value

A data frame containing `N` rows and 2 columns which represent `xi1` and the log-posterior density (`lpost`)

See Also

- `mcmc_mix` for MCMC for the discrete extreme value mixture distribution.
Smix

Survival function of discrete extreme value mixture distribution

Description

Smix returns the survival function at x for the discrete extreme value mixture distribution.

Usage

Smix(x, u, xi1, xi2, sig, geo, phi, log = FALSE)

Arguments

x         Vector of positive integers
u         Scalar, positive integer threshold
xi1       Scalar, shape parameter for values below or equal to u
xi2       Scalar, shape parameter of integer generalised Pareto distribution (IGPD), for
          values above u
sig       Scalar, scale parameter of IGPD, for values above u
geo       Boolean. If 'TRUE', the geometric distribution is used for the values below u.
          If 'FALSE', the discrete power law is used.
phi       Scalar, exceedance probability of u, between 0.0 and 1.0 exclusive
log       Boolean (default 'FALSE'), whether the survival function should be returned on
          the log scale.

Value

A numeric vector of the same length as x

See Also

dmix for the corresponding probability mass function, Supp for the survival function of the discrete
power law.

Examples

Smix(10:15, 12, 2.0, 0.5, 1.0, TRUE, 0.2)
Smix(10:15, 12, 2.0, 0.5, 1.0, FALSE, 0.2)
Supp

Survival function of discrete power law

Description

Supp returns the survival function at x for the discrete power law with exponent \((1.0 / xi1 + 1.0)\), for values greater than or equal to u.

Usage

Supp(x, u, xi1, log = FALSE)

Arguments

x Vector of positive integers
u Scalar, non-negative integer threshold
xi1 Scalar, a positive real number representing the shape parameter
log Boolean (default 'FALSE'), whether the survival function should be returned on the log scale.

Details

The survival function used is \(S(x) = Pr(X >= x)\), where X is a random variable following the discrete power law. The inclusion of x in the sum means \(S(x)\) may not necessarily equal to \(Pr(X > x)\) as the distribution is discrete. In the case of discrete power law, it can be shown that \(S(x) = hzeta(alpha, x)/hzeta(alpha, u)\), where \(hzeta\) is the Hurwitz zeta function i.e. \(hzeta(y, z) = z^(-y) + (z+1)^(-y) + (z+2)^(-y) + ...\) and \(alpha = 1.0 / xi1 + 1.0\). That \(xi1\) is used instead of \(alpha\) is for alignment with the parametrisation in \(dmix\), \(Smix\) and \(mcmc_mix\).

Value

A numeric vector of the same length as x

See Also

dupp for the corresponding probability mass function, Smix for the survival function of the discrete extreme value mixture distribution.

Examples

Supp(c(10, 20, 30, 40, 50), 12, 2.0)
topo_sort_kahn  

Return a sorted vector of nodes id

Description

Return a sorted vector of nodes id

Usage

topo_sort_kahn(g, random = FALSE)

Arguments

g An igraph object of a DAG
random Boolean, whether the order of selected nodes is randomised in the process

Value

A data frame with two columns: "id" is the names of nodes in g, and "id_num" is the topological ordering

Examples

df0 <- data.frame(from = c("a", "b"), to = c("b", "c"), stringsAsFactors = FALSE)
g0 <- igraph::graph_from_data_frame(df0, directed = TRUE)
topo_sort_kahn(g0)
Index

* datasets
  - chi_citations, 2
  - cran_dependencies, 3

chi_citations, 2, 3
cran_dependencies, 2, 3

df_to_graph, 3, 8
dmix, 4, 6, 12
dupp, 5, 5, 13

ggetf, dep, 6
getf, dep (getf, dep, df), 7
getf, dep_all (getf, dep, df), 7
getf, dep_all_packages, 6, 7, 8
getf, dep, df, 7
getf, graph_all_packages, 6, 7, 8

mcmc_mix, 9, 11
mcmc_upp, 10, 10

Smix, 5, 12, 13
Supp, 6, 12, 13

topo_sort_kahn, 14