## Package ‘skm’

October 14, 2022

**Type** Package  
**Title** Selective k-Means  
**Version** 0.1.5.4  
**Author** Guang Yang  
**Maintainer** Guang Yang &lt;gyang274@gmail.com&gt;  

**Description**  
Algorithms for solving selective k-means problem, which is defined as finding k rows in an m x n matrix such that the sum of each column minimal is minimized.  
In the scenario when m == n and each cell value in matrix is a valid distance metric, this is equivalent to a k-means problem. The selective k-means extends the k-means problem in the sense that it is possible to have m != n, often the case m < n which implies the search is limited within a small subset of rows. Also, the selective k-means extends the k-means problem in the sense that the instance in row set can be instance not seen in the column set, e.g., select 2 from 3 internet service provider (row) for 5 houses (column) such that minimize the overall cost (cell value) - overall cost is the sum of the column minimal of the selected 2 service provider.

**License** MIT + file LICENSE  
**LazyData** TRUE  

**URL** [http://github.com/gyang274/skm](http://github.com/gyang274/skm)  
**BugReports** [http://github.com/gyang274/skm/issues](http://github.com/gyang274/skm/issues)  
**RoxygenNote** 5.0.1  
**Depends** R (&gt;= 3.0.0), magrittr, data.table  
**Imports** methods, plyr, Rcpp (&gt;= 0.12.5), RcppParallel  
**LinkingTo** Rcpp, RcppArmadillo, RcppParallel  
**SystemRequirements** GNU make  
**Suggests** knitr, rmarkdown  
**VignetteBuilder** knitr
NeedsCompilation: yes
Repository: CRAN
Date/Publication: 2017-01-23 08:22:33

R topics documented:

```r
col_max_idx .................................................. 2
col_max_val .................................................... 3
col_min_idx .................................................... 4
col_min_val .................................................... 4
col_rgn_val ..................................................... 5
dist_wlatlng .................................................... 5
dist_wlatlng_cpp ................................................ 6
skmRpl_mlp_cpp .................................................. 7
skmSolution ...................................................... 8
skm_gdp_cpp ...................................................... 9
skm_minmax_cpp .................................................. 9
skm_mlp_cpp ..................................................... 10
skm_mls .......................................................... 11
skm_mls_cpp ..................................................... 12
skm_rgi_cpp ..................................................... 13
skm_rgs_cpp ..................................................... 14
skm_sgl_cpp ..................................................... 14
source_zip_list ................................................ 15
stratified_sampling ............................................ 16
zip2012 .......................................................... 16
```

Index 18

<table>
<thead>
<tr>
<th>col_max_idx</th>
<th>col_max_idx</th>
</tr>
</thead>
</table>

Description
calculate colvec max value index within limited range

Usage
col_max_idx(u, wlmt)

Arguments

<table>
<thead>
<tr>
<th>u</th>
<th>u: a numeric colvec</th>
</tr>
</thead>
<tbody>
<tr>
<td>wlmt</td>
<td>wlmt: limit search on colvec on indices within wlmt</td>
</tr>
</tbody>
</table>
**col_max_val**

**Value**

id an index of max value in u within wlmt w.r.t to original index

**Note**

cpp use index start from 0 vs r use index start from 1
in case of equal std:min/std:max take first index seen

**See Also**

Other matrix_minmax: *col_max_val, col_min_idx, col_min_val, col_rgn_val*

---

**col_max_val**

**Description**

calculate colvec max value within limited range

**Usage**

```
col_max_val(u, wlmt)
```

**Arguments**

- **u**: a numeric colvec
- **wlmt**: limit search on colvec on indices within wlmt

**Value**

vd min value in u within wlmt w.r.t to original index

**See Also**

Other matrix_minmax: *col_max_idx, col_min_idx, col_min_val, col_rgn_val*
**col_min_idx**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calculate colvec min value index within limited range</td>
</tr>
</tbody>
</table>

**Usage**

```r
col_min_idx(u, wlmt)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>u: a numeric colvec</td>
</tr>
<tr>
<td>wlmt</td>
<td>wlmt: limit search on colvec on indices within wlmt</td>
</tr>
</tbody>
</table>

**Value**

id an index of min value in u within wlmt w.r.t to original index

**Note**

cpp use index start from 0 vs r use index start from 1

in case of equal std:min/std:max take first index seen

**See Also**

Other matrix_minmax: `col_max_idx`, `col_max_val`, `col_min_val`, `col_rgn_val`

---

**col_min_val**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calculate colvec min value within limited range</td>
</tr>
</tbody>
</table>

**Usage**

```r
col_min_val(u, wlmt)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>u: a numeric colvec</td>
</tr>
<tr>
<td>wlmt</td>
<td>wlmt: limit search on colvec on indices within wlmt</td>
</tr>
</tbody>
</table>
**Description**
calculate colvec range = max - min value within limited range

**Usage**
col_rgn_val(u, wlmt)

**Arguments**
- **u**: a numeric colvec
- **wlmt**: limit search on colvec on indices within wlmt

**Value**
vd max - min value in u within wlmt w.r.t to original index

**See Also**
Other matrix_minmax: col_max_idx, col_max_val, col_min_idx, col_rgn_val

**Description**
calculate distance between coordinate1<lat1, lng1> and coordinate2<lat2, lng2>

**Usage**
dist_wlatlng(.lat1, .lng1, .lat2, .lng2, .measure = "mi")
Arguments

.lat1 latitude of coordinate1
.lng1 longitude of coordinate1
.lat2 latitude of coordinate2
.lng2 longitude of coordinate2
.measure - mi or km

Details

calculate the great circle distance between 2 points with Haversine formula, which deliberately ignores elevation differences.


dlon = lon2 - lon1

dlat = lat2 - lat1

\[
a = \sin^2\left(\frac{dlat}{2}\right) + \cos(\text{lat1}) \times \cos(\text{lat2}) \times \sin^2\left(\frac{dlon}{2}\right)
\]

\[
c = 2 \times \arcsin(\min(1, \sqrt{a}))
\]

\[
d = R \times c
\]

Description

calculate distance between coordinate1<lat1, lng1> and coordinate2<lat2, lng2>

Usage

dist_wlatlng_mi_cpp(lat1, lng1, lat2, lng2)

dist_wlatlng_km_cpp(lat1, lng1, lat2, lng2)

distSgl_wlatlng_cpp(lat1, lng1, lat2, lng2, measure = "mi")

distRpl_wlatlng_cpp(lat1, lng1, lat2, lng2, measure = "mi", distRpl_GS = 100L)

Arguments

lat1 latitude of coordinate1
lng1 longitude of coordinate1
lat2 latitude of coordinate2
lng2 longitude of coordinate2
The grain size of a parallel algorithm sets a minimum chunk size for parallelization. In other words, at what point to stop processing input on separate threads (as sometimes creating more threads can degrade the performance of an algorithm by introducing excessive synchronization overhead). Default is 100.

Details

calculate the great circle distance between 2 points with Haversine formula, which deliberately ignores elevation differences.


\[
dlon = \text{lon}_2 - \text{lon}_1 \\
dlat = \text{lat}_2 - \text{lat}_1 \\
a = \sin^2\left(\frac{dlat}{2}\right) + \cos(\text{lat}_1) \times \cos(\text{lat}_2) \times \sin^2\left(\frac{dlon}{2}\right) \\
c = 2 \times \arcsin\left(\min(1, \sqrt{a})\right) \\
d = R \times c
\]

dist_wlatlng_mi_cpp:
calculate distance between coordinate1<lat1, lng1> and coordinate2<lat2, lng2> in mile

dist_wlatlng_km_cpp:
calculate distance between coordinate1<lat1, lng1> and coordinate2<lat2, lng2> in kilometer

distSgl_wlatlng_cpp:
calculate distance between coordinate1<lat1, lng1> and coordinate2<lat2, lng2> in mile (measure = "mi") or kilometer (measure = "km"), default is mile.

implement as serial computing over vector of lat1, lng1, lat2, lng2

distRpl_wlatlng_cpp:
calculate distance between coordinate1<lat1, lng1> and coordinate2<lat2, lng2> in mile (measure = "mi") or kilometer (measure = "km"), default is mile.

implement as parallel computing over vector of lat1, lng1, lat2, lng2 via RcppParallel

Description

solve skm with multiple runs in parallel

Usage

skmRpl_mlp_cpp(x, k, s_must, max_it, max_at, skmRpl_GS = 100L)
Arguments

- **x**: an m x n matrix often m < n, as a convention index rows of x with s, and cols of x with t so x(i, j) can be expressed as (s_i, t_j) equally.
- **k**: number of index to be selected from x row index start from 0.
- **s_must**: an index vector set should be selected before selecting other index.
- **max_it**: max number of iterations can run for optimizing result. max number of iterations within a single initial run on optimal path.
- **max_at**: max number of attempts or repeats on running for optimal results, max number of random initialization for finding optimal results.
- **skmRpl_GS**: skmRpl_GS: RcppParallel grain size when run skmRpl_mlp_cpp

Details

refer skm_sgl_cpp

Value

skmSolution skmSolution present in r list

Description

class skmSolution, which often returned via skm solver implemented in cpp

Usage

skmSolution

Format

An object of class C++Class of length 1.

Details

an skmSolution instance has two member variable:

- **o**: objective sum(min(x.subview(i in s, all j), min over all i), sum over all j)
- **s**: selected index set of row index start from 0
**skm_gdp_cpp**

**Description**

solve selective kmeans via a greedy propagation.

**Usage**

```r
skm_gdp_cpp(x, k = 0L)
```

**Arguments**

- `x`: an m x n matrix of s - t - dist
- `k`: number of index to be selected from x row index start from 0.

**Details**

skm_gdp_cpp init with an input m x n matrix x and want to select an index set s of size k from x row index started from 0 such that

minimize \( \text{sum}(\text{min}(x_{\text{subview}(i \in s, \text{all } j)}, \text{min over all } i), \text{sum over all } j) \)

skm_gdp_cpp solve the problem with greedy propagation via selecting the current best addon index from the index set left, addon index is defined as such index when addon to the selected one can bring the most improvement.

since skm_gdp_cpp would select index one by one, and no return, e.g., if select index A for k = 1, then selection on k = 2 would build on k = 1, so index A is always present in the solution, so all index can be ranked w.r.t when it would be considered as the best addon. as a result skm_gdp_cpp a parameter k is not always required, so default k = 0 will return a vector of size m, and user can select to top k as solution for k.

**Value**

- `s`: a ranked index 0 - m - 1 where the top k would minimize \( \text{sum}(\text{min}(x_{\text{subview}(i \in s(0..k-1), \text{all } j)}, \text{min over all } i), \text{sum over all } j) \)

**skm_minmax_cpp**

**Description**

skm via min-max on in cpp - subroutine of skm_sgl_cpp calls

**Usage**

```r
skm_minmax_cpp(x, s_must)
```

**Arguments**

- `x`: an m x n matrix of s - t - dist
- `s_must`: number of index to be selected from x row index start from 0.
Arguments

\[
x \quad \text{an } m \times n \text{ matrix often } m > n
\]
\[
s_{\text{must}} \quad \text{matrix } x \text{ row index start from 0 that must be selected with priority}
\]

Details

skm_minmax_cpp init an input \( m \times n \) matrix \( x \), and a priority vector \( s_{\text{must}} \) would select \( n \) indicies from \( m \) such that:

\[
\text{minimize } \sum(\text{min}(x(i, j) \text{ where } i < 1..n> \text{ and } j < 1..n> \text{ each use } <1..n> \text{ once}))
\]

so in case \( m \leq n \) it simply select all \( m \) - should always be apply on matrix with \( m > n \) - it is designed as a expectation step in skm_cpp on updating \( s \).

it select \( i \) in \( <1..m> \) such that \( i \) has the colwise_min_idx on column \( j \) where \( j \) has max difference of (colwise_max_val - colwise_min_val), it then remove row \( i \) col \( j \) from matrix and repeat.

\( s_{\text{must}} \) presents the indices with priority so that the selection must select first indicies within \( s_{\text{must}} \) and then select other indicies outside \( s_{\text{must}} \).

an example skm_minmax_cpp is superior in bound worst case compare to greedy: \( x = [1 100; 4 200; 9 900] \): greedy 1 then 200, min-max 100 then 2, and greedy give [1 100; 4 200] with 201 and minmax give [1 100; 2 400] with 102.

Description

solve skm with multiple runs in serial and return all w. optim

Usage

skm_mlp_cpp(x, k, s_must, max_it, max_at)

Arguments

\[
x \quad \text{an } m \times n \text{ matrix often } m < n, \text{ as a convention index rows of } x \text{ with } s, \text{ and cols of } x \text{ with } t \text{ so } x(i, j) \text{ can be expressed as } (s_i, t_j) \text{ equally.}
\]
\[
k \quad \text{number of index to be selected from } x \text{ row index start from 0.}
\]
\[
s_{\text{must}} \quad \text{an index vector set should be selected before selecting other index.}
\]
\[
\text{max_it} \quad \text{max number of iterations can run for optimizing result. max number of iterations within a single initial run on optimal path.}
\]
\[
\text{max_at} \quad \text{max number of attempts or repeats on running for optimal results, max number of random initialization for finding optimal results.}
\]

Details

refer skm_sgl_cpp
**skm_mls**

**Value**

skmSolution skmSolution present in r list

**See Also**

Other skm: skm_mls_cpp, skm_rgi_cpp, skm_rgs_cpp, skm_sgl_cpp

---

**Description**

a selective k-means problem solver - wrapper over skm_mls_cpp

**Usage**

skm_mls(x, k = 1L, s_colname = "s", t_colname = "t", d_colname = "d",
w_colname = NULL, s_ggrp = integer(0L), s_must = integer(0L),
max_it = 100L, max_at = 100L, auto_create_ggrp = TRUE,
extra_immaculatism = TRUE, extra_at = 10L)

**Arguments**

- **x**: data.table with s - t - d(s, t): s<source> - t<target> - d<distance> where s<source> and t<target> must characters and d<distance> must numeric. aware d<distance> is not necessary as an euclidean or any distance and even necessary as symmetric - d(s, t) can be unequal to d(t, s) - view d as such a measure of the cost of assigning one to the other!
- **k**: number of centers
- **s_colname**: s<source>
- **t_colname**: t<target>
- **d_colname**: d<distance> - view d as cost of assigning t into s. also modify the input data or build in the algorithm can solve problem with a different fixed cost on using each s as source - i prefer to moddify data so that the algorithm is clean and clear - i will show a how to in vignette
- **w_colname**: w<weighting> - optional: when not null will optimize toward objective to minimize d = d * w such as weighted cost of assigning t into s
- **s_ggrp**: s_init will be stratified sampling from s w.r.t s_ggrp.
- **s_must**: length <= k-1 s must in result: conditional optimizing.
- **max_it**: max number of iterations can run for optimizing result.
- **max_at**: max number of attempts/repeats on running for optimal.
- **auto_create_ggrp**: boolean indicator of whether auto creating the group structure using the first letter of s when s_ggrp is integer(0).
extra_immaculatism
boolean indicator of whether making extra runs for improving result consistency when multiple successive k is specified, e.g., k = c(9L, 10L).

extra_at
an integer specifying the number of extra runs when argument extra_immaculatism is TRUE.

Details
a selective k-means problem is defined as finding a subset of k rows from a m x n matrix such that the sum of each column minimal is minimized.
skm_mls would take data.table (data.frame) as inputs, rather than a matrix, assume that a data.table of s - t - d(s, t) for all combination of s and t, choose k of s that minimizes sum(min(d(s, t) over selected k of s) over t).

Value
data.table
 o - objective - based on d_colname
 w - weighting - based on w_colname
 k - k<list> - based on k - input
 s - s<source> - based on s_colname
 d - weighed averge value of d_colname weighed by w_column when s are selected.

Description
solve skm with multiple runs in serial and return all w. optim and s_init stratified sampled w.r.t g

Usage
skm_mls_cpp(x, k, g, s_must, max_it, max_at)

Arguments
x
an m x n matrix often m < n, as a convention index rows of x with s, and cols of x with t so x(i, j) can be expressed as (s_i, t_j) equally.
k
number of index to be selected from x row index start from 0.
g
stratify structure, often info on grouping of v so that algorithm should make random initialization from stratified sample across groups.
s_must
an index vector set should be selected before selecting other index.
max_it
max number of iterations can run for optimizing result. max number of iterations within a single initial run on optimal path.
max_at
max number of attempts or repeats on running for optimial results, max number of random initialization for finding optimial results.
skm_rgi_cpp

Details
refer skm_sgl_cpp

Value
skmSolution skmSolution present in r list

See Also
Other skm: skm_mlp_cpp, skm_rgi_cpp, skm_rgs_cpp, skm_sgl_cpp

skm_rgi_cpp skm_rgi_cpp

Description
solve skm with single and random size k s_init

Usage
skm_rgi_cpp(x, k, s_must, max_it)

Arguments
x an m x n matrix often m < n, as a convention index rows of x with s, and cols of
x with t so x(i, j) can be expressed as (s_i, t_j) equally.
k number of index to be selected from x row index start from 0.
s_must an index vector set should be selected before selecting other index.
max_it max number of iterations can run for optimizing result. max number of iterations
within a single initial run on optimal path.

Details
refer skm_sgl_cpp

Value
skmSolution

See Also
Other skm: skm_mlp_cpp, skm_mls_cpp, skm_rgs_cpp, skm_sgl_cpp
skm_rgs_cpp

Description
solve skm with single and random size k s_init stratified sampled w.r.t g

Usage
skm_rgs_cpp(x, k, g, s_must, max_it)

Arguments
x an m x n matrix often m < n, as a convention index rows of x with s, and cols of x with t so x(i, j) can be expressed as (s_i, t_j) equally.
k number of index to be selected from x row index start from 0.
g stratify structure, often info on grouping of v so that algorithm should make random initialization from stratified sample across groups.
s_must an index vector set should be selected before selecting other index.
max_it max number of iterations can run for optimizing result. max number of iterations within a single initial run on optimal path.

Details
refer skm_sgl_cpp

Value
skmSolution

See Also
Other skm: skm_mlp_cpp, skm_mls_cpp, skm_rgi_cpp, skm_sgl_cpp

skm_sgl_cpp

Description
solve skm with single and a fixed given s_init

Usage
skm_sgl_cpp(x, s_init, s_must, max_it)
### Arguments

- **x**
  - An m x n matrix often m < n, as a convention index rows of x with s, and cols of x with t so x(i, j) can be expressed as (s_i, t_j) equally.

- **s_init**
  - An init vector of k index to start the search of optimal index set of k, length of s_init also defined the number of index want to be select.

- **s_must**
  - An index vector set should be selected before selecting other index.

- **max_it**
  - Max number of iterations can run for optimizing result. Max number of iterations within a single initial run on optimal path.

### Details

A numeric m x n matrix x often m « n and want to select a subset of k from m such that it minimize the sum(min(x(i, j) - minimum w.r.t each j over all i within selected index set), over all i)

If m == n and x(i, j) as euclidean distance then it is equivalent to kmeans

skm can select a combined set for deploying resource, for example, where to build 5 warehouses on united states, which often different than build these warehouses via select the current best one by one.

### Value

- **skmSolution**

### See Also

Other skm: `skm_mlp_cpp`, `skm_mls_cpp`, `skm_rgi_cpp`, `skm_rgs_cpp`

### Description

A list of zip code used in skm package demonstration.

### Usage

`source_zip_list`

### Format

A character vector of length 51 includes one 5 digits zip code selected from each state, where the most central zip code in each state selected.
**stratified_sampling**  

**Description**  

select k elements from vector v w.r.t stratify structure group g. TODO - implementing via template so v is flexible as vec or uvec.

**Usage**  

```r
stratified_sampling(v, k, g)
```

**Arguments**  

- **v** - <vector> v: a numeric candidate v from which draw sample.
- **k** - <integer> k: selection sample size.
- **g** - <vector> g: stratify structure g - info on grouping of v so that the selected sample is stratified across groups.

**Value**  

- **s** - <vector> s: a vector select from v length k stratified by g.

**Note**  

v is required as an integer vector for using in skm

---

**zip2012**  

**Description**  

a zip code database with latitude, longitude, population and income.

**Usage**  

```r
zip2012
```
Format

A data table with 28844 rows and 9 variables:

- **zip**  zip code, 5 digits zip code in U.S.
- **lat**  latitude
- **lng**  longitude
- **pop**  population
- **ink**  income
- **city**  city
- **state**  state
- **p_pop**  percentage of population w.r.t total population
- **p_ink**  percentage of income w.r.t total income

Source

http://federalgovernmentzipcodes.us/
Index

* datasets
  skmSolution, 8
  source_zip_list, 15
  zip2012, 16
  col_max_idx, 2, 3–5
  col_max_val, 3, 3, 4, 5
  col_min_idx, 3, 4, 5
  col_min_val, 3, 4, 5
  col_rgn_val, 3–5, 5
  dist_wlatlng, 5
  dist_wlatlng_cpp, 6
  dist_wlatlng_km_cpp (dist_wlatlng_cpp), 6
  dist_wlatlng_mi_cpp (dist_wlatlng_cpp), 6
  distRpl_wlatlng_cpp (dist_wlatlng_cpp), 6
  distSgl_wlatlng_cpp (dist_wlatlng_cpp), 6
  Rcpp_skmSolution (skmSolution), 8
  Rcpp_skmSolution-class (skmSolution), 8
  skm_gdp_cpp, 9
  skm_minmax_cpp, 9
  skm_mlp_cpp, 10, 13–15
  skm_mls, 11
  skm_mls_cpp, 11, 12, 13–15
  skm_rgi_cpp, 11, 13, 14, 15
  skm_rgs_cpp, 11, 13, 14, 15
  skm_sgl_cpp, 11, 13, 14, 14
  skmRpl_mlp_cpp, 7
  skmSolution, 8
  source_zip_list, 15
  stratified_sampling, 16
  zip2012, 16