Package 'tlsh'

November 6, 2020

Type Package

Title Transitive Locality-Sensitive Hashing (LSH) for Record Linkage

Version 0.1.0

Depends R (>= 3.5.0), blink, stats, utils, plyr, igraph, bit64

Imports

Suggests knitr, ggplot2, rmarkdown

VignetteBuilder knitr

Description An implementation of the blocking algorithm transitive locality-sensitive hashing (TLSH) in Steorts, Ventura, Sadinle, Fienberg (2014) <DOI:10.1007/978-3-319-11257-2_20>, which is a k-means variant of locality sensitive hashing. The method is illustrated with examples and a vignette.

Encoding UTF-8

LazyData true

License GPL-3

RoxygenNote 7.1.1.9000

NeedsCompilation no

Author Rebecca Steorts [aut, cre]

Maintainer Rebecca Steorts <beka@stat.duke.edu>

Repository CRAN

Date/Publication 2020-11-06 17:00:02 UTC

R topics documented:

`block.ids.from.blocking` *Returns the block ids associated with a blocking method.*

Description

Returns the block ids associated with a blocking method.

Usage

```
block.ids.from.blocking(blocking)
```

Arguments

`blocking`	A list of the blocks.

Value

A list of the blocks ids that corresponds to each block

Examples

```
tlsh.blocks <- block_setup_v2(r.set = RLdata500[1:250,c(-2,-4)], b=10, save_signature=FALSE, k=1)
block.ids.from.blocking(tlsh.blocks)
```

`block_setup_v2` *Function that divides all records into bins using locality sensitive hashing and using TLSH (based upon community detection technique)*

Description

import blink

Usage

```
block_setup_v2(r.set, b = 22, save_signature = FALSE, k = 5)
```

Arguments

r.set	Record set (shingled records)
b	Band
save_signature	Flag of whether or not to save the signature
k	Shingle size

Value

List of blocks where a particular index is the record id in the original data set

Examples

```
r.set <- RLdata500[1:3,c(-2)]
block_setup_v2(r.set = RLdata500[1:3,c(-2)], b=22, save_signature=FALSE, k=2)
```

---

compare_buckets *Function that creates a similarity graph and divides it into communities (or blocks) for entity resolution*

Description

Function that creates a similarity graph and divides it into communities (or blocks) for entity resolution

Usage

```
compare_buckets(hashed_signatures, max_bucket_size = 1000)
```

Arguments

hashed_signatures	The hashed signatures
max_bucket_size	The largest block size allowed by user

Value

max_bucket_size The largest bucket size (or block size) that one can handle

Examples

```
head(data <- RLdata500[-c(2,4)])
minidata <- data[1:2,]
head(all_the_shingles <- apply(minidata,1,shingles,k=8))
head(minhash.minidata <- minhash_v2(all_the_shingles, p=10))
hashed_signature <- hash_signature(minhash.minidata, b=5)
compare_buckets(hashed_signature, max_bucket_size=200)
```

`confusion.from.blocking` *Perform evaluations (recall) for blocking.*

Description

Perform evaluations (recall) for blocking.

Usage

```
confusion.from.blocking(blocking, true_ids, recall.only = FALSE)
```

Arguments

`blocking`	A list of the blocks
`true_ids`	The true identifiers for comparisons
`recall.only`	Flag that when true only prints the recall, otherwise prints many evaluation metrics in a list

Value

A vector of that returns the recall and the precision

Examples

```
r.set <- RLdata500[1:250,c(-2)]
tlsh.blocks <- block_setup_v2(r.set, b=22, save_signature=FALSE, k=2)
confusion.from.blocking(tlsh.blocks, identity.RLdata500, recall.only=TRUE)
```

`eval.blocksetup` *Function to evaluate the blocking step*

Description

import blink

Usage

```
eval.blocksetup(dat, k = 5, b = 21, key)
```

Arguments

`dat`	Data set
`k`	Parameter k, which is the number of shingle, tokens, or grams to break the string into
`b`	Number of buckets
`key`	Unique identifier

Value

Recall and runtime

Examples

```
r.set <- RLdata500[1:50,c(-2)]
eval.blocksetup(r.set, k=2, b=22, key=identity.RLdata500)
```

`extract_pairs_from_band` *Function that extracts pairs of records from a band in the signature matrix M import bit64*

Description

Function that extracts pairs of records from a band in the signature matrix M import bit64

Usage

```
extract_pairs_from_band(a_band)
```

Arguments

a_band	Band of the signature matrix M

Value

The edgelist of record pairs that are connected

Examples

```
band1 <- c(2,1,2,1,2)
extract_pairs_from_band(band1)
band2 <- c(6,7,8,9,6)
extract_pairs_from_band(band2)
band.12 <- rbind(band1, band2)
apply(band.12,1,extract_pairs_from_band)
```

`hash_signature` *Function to take a signature matrix M composed of b bands and r rows and return a bucket for each band for each record*

Description

Function to take a signature matrix M composed of b bands and r rows and return a bucket for each band for each record

Usage

```
hash_signature(signature, b)
```

Arguments

`signature`	Signature matrix M composed of b bands and r rows
`b`	Number of bands

Value

Bucket for each band for each record

Examples

```
head(data <- RLdata500[-c(2,4)])
minidata <- data[1:2,]
head(all_the_shingles <- apply(minidata,1,shingles,k=8))
head(minhash.minidata <- minhash_v2(all_the_shingles, p=10))
hash_signature(minhash.minidata, b=2)
hash_signature(minhash.minidata, b=5)
```

`minhash_v2` *Function to create a matrix of minhashed signatures*

Description

Function to create a matrix of minhashed signatures

Usage

```
minhash_v2(
  shingled_records,
  p,
  do_one_hash_and_record = do_one_hash_and_record
)
```

Arguments

shingled_records	Shingled records
p	Number of permutations to be applied to the hash function
do_one_hash_and_record	Combination of one hash and one record

Value

Computes an integer-valued matrix of minhash signatures with one row per permutation and one column per record

Examples

```
head(data <- RLdata500[-c(2,4)])
minidata <- data[1:2,]
head(all_the_shingles <- apply(minidata,1,shingles,k=8))
head(minhash.minidata <- minhash_v2(all_the_shingles, p=10))
```

---

my_hash *Function that applies a hash function to each column of the band from the signature matrix import bit64*

Description

Function that applies a hash function to each column of the band from the signature matrix import bit64

Usage

```
my_hash(a_band)
```

Arguments

a_band	Band from the signature matrix M

Value

a 64 bit integer

Examples

```
band1 <- c(2,1,2,1,2)
band2 <- c(4,5,2,1,9)
combined_band <- rbind(band1,band2)
my_hash(combined_band)
```

primest *Function to generate all primes larger than an integer n1 (lower limit) and less than any other integer n2 (upper limit)*

Description

Function to generate all primes larger than an integer n1 (lower limit) and less than any other integer n2 (upper limit)

Usage

```
primest(n1 = 1, n2)
```

Arguments

n1	An integer taken to be 1 as the default
n2	Any integer n2

Value

Generates all prime numbers with the above constraints

Examples

```
primest(1, 5)
primest(1, 17)
```

reduction.ratio *Returns the reduction ratio associated with a blocking method*

Description

Returns the reduction ratio associated with a blocking method

Usage

```
reduction.ratio(block.labels)
```

Arguments

block.labels	A list of the blocks labels.

Value

The reduction ratio

Examples

```
tlsh.blocks <- block_setup_v2(r.set = RLdata500[1:50,c(-2)], b=22, save_signature=FALSE, k=2)
block.ids <- block.ids.from.blocking(tlsh.blocks)
reduction.ratio(block.ids)
```

---

reduction.ratio.from.blocking

Returns the reduction ratio associated with a blocking method

---

Description

Returns the reduction ratio associated with a blocking method

Usage

```
reduction.ratio.from.blocking(blocking)
```

Arguments

blocking	The actual blocks

Value

The reduction ratio

Examples

```
tlsh.blocks <- block_setup_v2(r.set = RLdata500[1:50,c(-2,-4)], b=10, save_signature=FALSE, k=1)
reduction.ratio.from.blocking(tlsh.blocks)
```

---

rhash_funcs

Function to generate a vector of random hash functions (or optionally one vector-valued function)

Description

Function to generate a vector of random hash functions (or optionally one vector-valued function)

Usage

```
rhash_funcs(n, size, vector.valued, perfect = FALSE)
```

Arguments

n	Number of random hash functions
size	Range of each size
vector.valued	Flag for outputing vector of functions or vector-valued function
perfect	Flag for whether a perfect permutation should be done, or just a hash function

Value

Vector of n hash functions or a function which will take a number and return a vector of n different hashes of it

Examples

```
rhash_funcs(1, 1, vector.valued=FALSE, perfect=FALSE)
rhash_funcs(5, 1, vector.valued=FALSE, perfect=FALSE)
```

---

shingled_record_to_index_vec

Function to convert to tell what index the shingle corresponds to in the record

---

Description

Function to convert to tell what index the shingle corresponds to in the record

Usage

```
shingled_record_to_index_vec(shingled_record, universal_set)
```

Arguments

shingled_record	Shingled record
universal_set	Universal set of all shingles

Value

the index regarding where the shingle falls in the record

Examples

```
shingles("Alexander",2)
shingles("Alexander Smith", 2)
shingled_record_to_index_vec(shingles("Alexander",2), unique(shingles("Alexander Smith", 2)))
```

shingles *Function to shingle (token or gram) a string into its k components*

Description

Function to shingle (token or gram) a string into its k components

Usage

```
shingles(record, k)
```

Arguments

record	String or record
k	Parameter k, which is the number of shingle, tokens, or grams to break the string into

Value

Computes the shingled (tokened or grammed) version of a string

Examples

```
shingles("Alexander",2)
shingles("Alexander Smith", 2)
```

Index