Package 'TSTr'

October 31, 2015

Type Package

Title Ternary Search Tree for Auto-Completion and Spell Checking

Version 1.2

Date 2015-10-07

Author Ricardo Merino [aut, cre],
Samantha Fernandez [ctb]

Maintainer Ricardo Merino <ricardo.merino.raldua@gmail.com>

Description A ternary search tree is a type of prefix tree with up to three children and the ability for incremental string search. The package uses this ability for word auto-completion and spell checking. Includes a dataset with the 10001 most frequent English words.

License GPL-2

LazyData yes

Depends R (>= 3.2.0)

Imports stringr, stringdist, stats, data.table

Suggests knitr

VignetteBuilder knitr

NeedsCompilation no

R topics documented:

`TSTr-package` *Ternary Search Tree for Auto-Completion and Spell Checking*

Description

A ternary search tree is a type of prefix tree with up to three children and the ability for incremental string search. The package uses this ability for word auto-completion and spell checking. Includes a dataset with the 10001 most frequent English words.

Details

This package can be used to create a ternary search tree, more space efficient compared to standard prefix trees. Common applications for ternary search trees include spell-checking and auto-completion.

Author(s)

Ricardo Merino [aut, cre], Samantha Fernandez [ctb]

Maintainer: Ricardo Merino <ricardo.merino.raldua@gmail.com>

References

`https://en.wikipedia.org/wiki/Ternary_search_tree`

See Also

`newTree`

`addToTree` *Adds a set of strings to a ternary search tree*

Description

Updates a ternary search tree adding the words or strings from the input

Usage

```
addToTree(tree, input)
```

Arguments

`tree`	an existing ternary search tree to add words to.
`input`	a filepath to read from or a character vector containing the strings.

Details

Updates the tree and adds the words contained in a vector or a file. Reports each 10000 words it has added to the tree and takes around 30 sec. per 10k words on a 8Gb RAM computer. In addition, reports the number of words added and the total number of nodes when finished.

Value

An object of class ‘list‘ and ‘tstTree‘.

See Also

newTree

Examples

```
fruitTree <- newTree(c("apple", "orange"))
fruitTree <- addToTree(fruitTree, c("lemon", "pear"))
```

---

addWord *Adds a single word or string*

Description

Adds a single word to an existing ternary search tree

Usage

```
addWord(tree, string)
```

Arguments

`tree`	an existing ternary search tree to add words to.
`string`	a string of characters to be added to the tree.

Details

The string of characters is added to the existing tree.

Value

An object of class ‘list‘ and ‘tstTree‘.

See Also

newTree

Examples

```
fruitTree <- newTree(c("apple", "orange"))
fruitTree <- addWord(fruitTree, "lemon")
dimTree(fruitTree)
```

completeWord *Autocompletion of strings*

Description

Returns a character vector with the completed words

Usage

```
completeWord(tree, string)
```

Arguments

`tree`	an existing ternary search tree to search words in.
`string`	a string of characters to be completed.

Details

Searches recursively through the tree until all words starting with the specified string have been found, and stores them in a character vector.

Value

A vector of class 'character'.

Examples

```
fruitTree <- newTree(c("apple", "orange","apricot","cherry"))
fruits.ap <- completeWord(fruitTree, "ap")
fruits.ap
```

dimTree *Tree dimensions*

Description

Returns a numeric vector with the dimensions of the ternary search tree

Usage

```
dimTree(tree)
```

Arguments

`tree`	an existing ternary search tree.

Details

The first number in the vector is the number of words in the ternary search tree and the second is the number of nodes (each node is a character) in the tree.

Value

A numeric vector with the dimensions of the tree.

See Also

newTree

Examples

```
fruitTree <- newTree(c("apple", "orange","apricot","cherry"))
dimTree(fruitTree)
```

---

newTree *Creates a new ternary search tree*

Description

Creates a new ternary search tree containing the input words

Usage

```
newTree(input)
```

Arguments

`input`	a filepath to read from or a character vector containing the strings.

Details

Creates a new tree and adds the words contained in a vector or a file to the tree. Reports each 10000 words it has added to the tree and takes around 30 sec. per 10k words on a 8Gb RAM computer. In addition, reports the total number of words and nodes when finished.

Value

An object of class ‘list‘ and ‘tstTree‘.

See Also

addToTree

Examples

```
fruitTree <- newTree(c("apple", "orange"))
fileConn <- file("XMIwords.txt")
writeLines(head(XMIwords,100), fileConn)
close(fileConn)
enTree <- newTree("XMIwords.txt")
```

`PNcheck` *Spell checking using ternary search trees*

Description

Spell checking using TST and Peter Norvig's approach.

Usage

```
PNcheck(tree, string, useUpper = FALSE)
```

Arguments

`tree`	a ternary search tree containing the dictionary terms.
`string`	the misspelled string to correct.
`useUpper`	if TRUE, uppercase letters are also used to construct insertions and alterations of the string. Default is FALSE.

Details

The literature on spelling correction claims that around 80% of spelling errors are an edit distance of 1 from the target. For a word of length n, there will be n deletions, n-1 transpositions, 36n alterations, and 36(n+1) insertions, for a total of 74n+35 (of which a few are typically duplicates). PNcheck computes all these variations and search them in a ternary search tree.

For distance 2 the number of variations becomes (74n+35)^2 which makes PNcheck 3 orders of magnitude more expensive than SDcheck.

Value

A vector with the corrected words.

See Also

newTree

Examples

```
fruitTree <- newTree(c("Apple", "orange", "lemon"))
PNcheck(fruitTree,"lamon")
PNcheck(fruitTree,"apple", useUpper = TRUE)
```

`SDcheck` *Performs spell checking using symmetric delete spell correction*

Description

Spell checking for symmetric delete approach. Automatically detects the distance with which the keeper was pre-created.

Usage

```
SDcheck(keeper, string, summarize = FALSE)
```

Arguments

`keeper`	the structure used in the pre-calculation step to store the dictionary symmetrical deletions.
`string`	the misspelled string to correct.
`summarize`	if TRUE returns a list that summarizes the different distances of the corrected words. Default is FALSE.

Details

Generate terms with an edit distance <= maxdist (deletes only) from the query term. As the edit distance between two terms is symmetrical and the deletions from the dictionary terms have been pre-stored, the performance is three orders of magnitude better than Peter Norvig's approach for distance 2 and five orders for distance 3.

Value

A vector or a list with the corrected words.

See Also

`SDkeeper`

Examples

```
fruitTree <- SDkeeper(c("apple", "orange", "lemon"), 2)
SDcheck(fruitTree,"aple")
SDcheck(fruitTree,"aple", summarize = TRUE)
```

SDkeeper *Pre-creates a data.table or a ternary search tree*

Description

Pre-calculation step for symmetric delete spelling correction. Creates a data.table or a ternary search tree to store the dictionary symmetrical deletions.

Usage

```
SDkeeper(input, maxdist, useTST = FALSE)
```

Arguments

`input`	a filepath to read from or a character vector containing the strings from which to create the symmetrical deletions.
`maxdist`	the maximum distance to use for spell checking. The literature on spelling correction claims that around 80% of spelling errors are an edit distance of 1 from the target, and 99% an edit distance of 2. SDkeeper allows to use a distance between 1 and 3.
`useTST`	specifies if a TST must be used to store the symmetrical deletions. Default is FALSE, an indexed data.table will be used instead (better performance).

Details

Generates terms with an edit distance <= maxdist (deletes only) from each dictionary term and add them together with the original term to the dictionary. This has to be done only once during a pre-calculation step.

For a word of length n, an alphabet size of a, an edit distance of 1, there will be just n deletions, for a total of n terms at search time. This is three orders of magnitude less expensive (36 terms for n=9 and d=2) than Peter Norvig's approach, and language independent (the alphabet is not required to generate deletes). The cost of this approach is the pre-calculation time and storage space of x deletes for every original dictionary entry, which is acceptable in most cases.

Value

An object of class ‘data.table‘ or ‘tstTree‘ storing the symmetrical deletions of the specified distance.

See Also

`SDcheck`

Examples

```
fruitTree <- SDkeeper(c("apple", "orange", "lemon"), 2)
fruitTree <- SDkeeper(c("apple", "orange", "lemon"), 1, useTST = TRUE)
SDcheck(fruitTree,"aple")
```

searchWord *Search a string*

Description

Searches for a string within a ternary search tree

Usage

```
searchWord(tree, string)
```

Arguments

`tree`	an existing ternary search tree to search words in.
`string`	a string of characters to search for.

Details

Searches through the tree for the specified string. Returns TRUE if the string have been added to the tree and FALSE if not.

Value

logical. TRUE if the string is in the tree, FALSE if not.

Examples

```
fruitTree <- newTree(c("apple", "orange","apricot","cherry"))
searchWord(fruitTree, "apricot")
searchWord(fruitTree, "banana")
```

XMIwords *10001 most frequent English words*

Description

A character vector containing 10001 frequency-ordered English words. All words have 3 or more letters.

Format

The format is: chr [1:10001] "the" "and" "that" "for" "you" "with" "was" "this" "have" "but" "are" "not" "from" ...

References

Extracted from the English HC Corpora.

Examples

```
data(XMIwords)
str(XMIwords)
```

Index