

# Package: RPatternJoin (via r-universe)

October 27, 2024

**Type** Package

**Title** String Similarity Joins for Hamming and Levenshtein Distances

**Version** 1.0.0

**Date** 2024-10-11

**Description** This project is a tool for words edit similarity joins (a.k.a. all-pairs similarity search) under small ( $< 3$ ) edit distance constraints. It works for Levenshtein/Hamming distances and words from any alphabet. The software was originally developed for joining amino-acid/nucleotide sequences from Adaptive Immune Repertoires, where the number of words is relatively large ( $10^5$ - $10^6$ ) and the average length of words is relatively small (10-100).

**License** MIT + file LICENSE

**Suggests** Matrix, testthat, stringdist

**Imports** Rcpp ( $\geq 1.0.13$ ), stats

**LinkingTo** Rcpp, RcppArmadillo

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.2

**Language** en-US

**Repository** <https://matveevdaniil.r-universe.dev>

**RemoteUrl** <https://github.com/matveevdaniil/rpatternjoin>

**RemoteRef** HEAD

**RemoteSha** 8a10c4b2f349a76c680c72048f095f796617d897

## Contents

RPatternJoin-package . . . . .	2
edit_dist1_example . . . . .	3
similarityJoin . . . . .	4

<b>Index</b>	<b>7</b>
--------------	----------

---

RPatternJoin-package *String Similarity Joins for Hamming and Levenshtein Distances*

---

## Description

This project is a tool for words edit similarity joins under small ( $< 3$ ) edit distance constraints. It works for Levenshtein distance and Hamming (with allowed insertions/deletions to the end) distance.

## Details

The package offers several similarity join algorithms, all of which can be accessed through the [similarityJoin](#) function. The software was originally developed for edit similarity joins of short amino-acid/nucleotide sequences from Adaptive Immune Repertoires, where the number of words is relatively large ( $10^5 - 10^6$ ) and the average length of words is relatively small (10 – 100). The algorithms will work with any alphabet and any list of words, however, larger lists or word sizes can lead to memory issues.

## Author(s)

Daniil Matveev <dmavveev@sfsu.edu>

## See Also

[similarityJoin](#), [edit\\_dist1\\_example](#)

## Examples

```
library(RPatternJoin)

## Small example

similarityJoin(c("ABC", "AX", "QQQ"), 2, "Hamming", output_format = "adj_pairs")
#      [,1] [,2]
# [1,]   1   1
# [2,]   1   2
# [3,]   2   1
# [4,]   2   2
# [5,]   3   3

## Larger example

# The `edit_dist1_example` function generate a random list
# of `num_strings` strings with the average string length=`avg_len`.
strings <- edit_dist1_example(avg_len = 25, num_strings = 5000)

# Firstly let's do it with `stringdist` package.
```

```
library(stringdist)
unnamed(system.time({
  which(stringdist::stringdistmatrix(strings, strings, "lv") <= 1, arr.ind = TRUE)
})["elapsed"])
# Runtime on macOS machine with 2.2 GHz i7 processor and 16GB of DDR4 RAM:
# [1] 63.773

# Now let's do it with similarityJoin function.
unnamed(system.time({
  similarityJoin(strings, 1, "Levenshtein", output_format = "adj_pairs")
})["elapsed"])
# Runtime on the same machine:
# [1] 0.105
```

---

edit\_dist1\_example      *Generate Example Strings with Edit Distance 1*

---

## Description

This function generates a random list of `num_strings = 5n` strings such that each of `n` strings has one duplicate, one string with a deleted letter, one string with an inserted letter, and one string with a substituted letter.

## Usage

```
edit_dist1_example(avg_len = 25, num_strings = 5000)
```

## Arguments

<code>avg_len</code>	Average length of the strings.
<code>num_strings</code>	Number of strings to generate.

## Value

A character vector of generated strings.

## See Also

[similarityJoin](#)

---

similarityJoin      *Build Adjacency Matrix*

---

## Description

Build Adjacency Matrix

## Usage

```
similarityJoin(
  strings,
  cutoff,
  metric,
  method = "partition_pattern",
  drop_deg_one = FALSE,
  special_chars = TRUE,
  output_format = "adj_matrix"
)
```

## Arguments

strings	Input vector of strings. To avoid hidden errors, the function will give a warning if strings contain characters not in the English alphabet. To disable this warning, change <code>special_chars</code> to <code>FALSE</code> .
cutoff	Cutoff: 0,1,2. The function will search all pairs of strings with edit distance less than or equal to the cutoff.
metric	Edit distance type: Hamming, Levenshtein.
method	Method: <code>partition_pattern</code> , <code>semi_pattern</code> , <code>pattern</code> . This parameter determines what algorithm will be used for similarity join. Methods will differ in time and space complexity, but produce the same output. By default, we recommend using <code>partition_pattern</code> , since it is the most memory efficient.
drop_deg_one	Drop isolated strings: <code>TRUE</code> , <code>FALSE</code> . Works only for <code>output_format=adj_matrix</code> . The default is <code>FALSE</code> .
special_chars	Enable check for special characters in strings: <code>TRUE</code> , <code>FALSE</code> . The default is <code>TRUE</code> .
output_format	Output format: <code>adj_matrix</code> , <code>adj_pairs</code> . The default is <code>adj_matrix</code> .

## Value

If `output_format = adj_pairs` - 2-column matrix where each row is a pair of indices of strings with an edit distance  $\leq$  cutoff.

If `output_format = adj_matrix` - the same output is presented as a sparse adjacency matrix with corresponding strings and their indices in the original vector are stored in `dimnames` of the adjacency matrix.

I.e.  $(adj\_matrix[i, j]=1) \Leftrightarrow$  distance between `dimnames(adj_matrix)[[1]][i]` and `dimnames(adj_matrix)[[1]][j]` is  $\leq$  cutoff.

If `drop_deg_one` is `FALSE`, then `dimnames(adj_matrix)[[1]] = strings` and `dimnames(adj_matrix)[[2]] = 1:length(s)`. Otherwise, `dimnames(adj_matrix)[[1]] = strings` without isolated strings and `dimnames(adj_matrix)[[2]] = original indices of strings in dimnames(adj_matrix)[[1]]` (original = index in input strings vector).

### See Also

[edit\\_dist1\\_example](#)

### Examples

```
library(RPatternJoin)
library(Matrix)

## Example 1
# Consider the following example with small similar words:
strings <- c("cat", "ecast", "bat", "cats", "chat")
# Let's find all pairs s.t. strings can be modified
# to each other with at most 2 substitutions.
# For this we choose our metric to be Hamming distance and cutoff to be 2.
metric <- "Hamming"
cutoff <- 2
# By default we use 'partition_pattern' method
# since it is the most memory efficient.
method <- "partition_pattern"
# Let's output the result as an adjacency matrix.
output_format <- "adj_matrix"
drop_deg_one <- TRUE

similarityJoin(
  strings, cutoff, metric,
  method = method, drop_deg_one = drop_deg_one)
# 3 x 3 sparse Matrix of class "dgCMatrix"
#   cat bat cats
# 1  1  1  1
# 3  1  1  1
# 4  1  1  1

## Example 2
# On the same strings, let's calculate pairs of strings with edit distance  $\leq 1$ .
cutoff <- 1
metric <- "Levenshtein"
# Let's output the result as an adjacency matrix, but drop strings without any connections.
drop_deg_one <- FALSE

similarityJoin(
  strings, cutoff, metric,
  method = method, drop_deg_one = drop_deg_one)
#   cat ecast bat cats chat
# 1  1     .  1  1  1
# 2  .     1  .  .  .
# 3  1     .  1  .  .
```

```
# 4 1 . . 1 .
# 5 1 . . . 1

## Example 3
# Now let's simulate a larger example.

# The `edit_dist1_example` function generate a random list
# of `num_strings` strings with the average string length=`avg_len`.
strings <- edit_dist1_example(avg_len = 25, num_strings = 5000)

# Firstly let's do it with `stringdist` package.

library(stringdist)
system.time({
  which(stringdist::stringdistmatrix(strings, strings, "lv") <= 1, arr.ind = TRUE)
})["elapsed"]
# Runtime on macOS machine with 2.2 GHz i7 processor and 16GB of DDR4 RAM:
# elapsed
# 63.773

# Now let's do it with similarityJoin function.
system.time({
  similarityJoin(strings, 1, "Levenshtein", output_format = "adj_pairs")
})["elapsed"]
# Runtime on the same machine:
# elapsed
# 0.105
```

# Index

\* **package**

    RPatternJoin-package, [2](#)

edit\_dist1\_example, [2](#), [3](#), [5](#)

RPatternJoin (RPatternJoin-package), [2](#)

RPatternJoin-package, [2](#)

similarityJoin, [2](#), [3](#), [4](#)