

# Package ‘nlpembeds’

February 4, 2025

**Title** Natural Language Processing Embeddings

**Version** 1.0.0

**Description** Provides efficient methods to compute co-occurrence matrices, pointwise mutual information (PMI) and singular value decomposition (SVD). In the biomedical and clinical settings, one challenge is the huge size of databases, e.g. when analyzing data of millions of patients over tens of years. To address this, this package provides functions to efficiently compute monthly co-occurrence matrices, which is the computational bottleneck of the analysis, by using the 'RcppAlgos' package and sparse matrices. Furthermore, the functions can be called on 'SQL' databases, enabling the computation of co-occurrence matrices of tens of gigabytes of data, representing millions of patients over tens of years. Partly based on Hong C. (2021) <[doi:10.1038/s41746-021-00519-z](https://doi.org/10.1038/s41746-021-00519-z)>.

**Imports** data.table, magrittr, Matrix, methods, parallel, RcppAlgos, reshape2, RSQLite, rsvd

**Depends** R (>= 3.5.0)

**Suggests** knitr, testthat

**VignetteBuilder** knitr

**License** GPL-3

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**URL** <https://gitlab.com/thomaschl/nlpembeds>

**BugReports** <https://gitlab.com/thomaschl/nlpembeds/-/issues>

**NeedsCompilation** no

**Author** Thomas Charlon [aut, cre] (<<https://orcid.org/0000-0001-7497-0470>>),  
Doudou Zhou [ctb] (<<https://orcid.org/0000-0002-0830-2287>>),  
CELEHS [aut] (<<https://celehs.hms.harvard.edu>>)

**Maintainer** Thomas Charlon <[charlon@protonmail.com](mailto:charlon@protonmail.com)>

**Repository** CRAN

**Date/Publication** 2025-02-04 08:30:01 UTC

## Contents

build_df_cooc . . . . .	2
build_spm_cooc_sym . . . . .	3
get_pmi . . . . .	3
get_svd . . . . .	4
spm_to_df . . . . .	5
sql_cooc . . . . .	5
%<>% . . . . .	7
%%\$% . . . . .	8
%>% . . . . .	8
<b>Index</b>	<b>9</b>

---

build_df_cooc	<i>Compute monthly co-occurrence matrix</i>
---------------	---

---

### Description

Compute monthly co-occurrence matrix

### Usage

```
build_df_cooc(
  df_ehr,
  uniq_codes = NULL,
  n_cores = 1,
  min_code_freq = 5,
  gc_before_parallel = TRUE
)
```

### Arguments

df_ehr	Input data frame, monthly counts with columns Patient, Month, Parent_Code, Count
uniq_codes	Not required, useful for sql_cooc function
n_cores	Number of cores
min_code_freq	Filter matrix based on feature frequency
gc_before_parallel	Call garbage collector before computation

### Value

Co-occurrence sparse matrix

**Examples**

```
df_ehr = data.frame(Month = c(1, 1, 1, 2, 2, 3, 3, 4, 4),
                    Patient = c(1, 1, 2, 1, 2, 1, 1, 3, 4),
                    Parent_Code = c('C1', 'C2', 'C2', 'C1', 'C1', 'C1', 'C2',
                                     'C3', 'C4'),
                    Count = 1:9)

spm_cooc = build_df_cooc(df_ehr)
```

---

build\_spm\_cooc\_sym      *Build symmetric sparse matrix from data frame*

---

**Description**

Build symmetric sparse matrix from data frame

**Usage**

```
build_spm_cooc_sym(df_cooc)
```

**Arguments**

df\_cooc                  Symmetric sparse matrix in data frame format

**Value**

Matrix::sparseMatrix object, symmetric sparse matrix

---

get\_pmi                      *Compute pointwise mutual information (PMI)*

---

**Description**

Compute pointwise mutual information (PMI)

**Usage**

```
get_pmi(spm_cooc)
```

**Arguments**

spm\_cooc                  Co-occurrence sparse matrix, either a triangular sparse matrix or a dataframe

**Value**

PMI symmetric matrix

**Examples**

```
df_ehr = data.frame(Patient = c(1, 1, 2, 1, 2, 1, 1, 3, 4),
                    Month = c(1, 1, 1, 2, 2, 3, 3, 4, 4),
                    Parent_Code = c('C1', 'C2', 'C2', 'C1', 'C1', 'C1',
                                   'C2', 'C3', 'C4'),
                    Count = 1:9)

spm_cooc = build_df_cooc(df_ehr)

m_pmi = get_pmi(spm_cooc)
```

---

`get_svd`*Compute random singular value decomposition (rSVD)*

---

**Description**

Random SVD is an efficient approximation of truncated SVD, in which only the first principal components are returned. It is computed with the `rsvd` package, and the author suggests that the number of dimensions requested  $k$  should be:  $k < n / 4$ , where  $n$  is the number of features, for it to be efficient, and that otherwise one should rather use either SVD or truncated SVD. When computing SVD on PMI, we only want to use the singular values corresponding to the positive eigen values. We do not know beforehand how many we will have to filter out, so there is two parameters: `'embedding_dim'` for the requested output dimension, and `'svd_rank'` for the actual SVD computation, by default twice the requested dimension, and a warning may be thrown if `'svd_rank'` needs to be manually increased. Computation may be expensive and manually optimizing the `'svd_rank'` parameter might save significant time.

**Usage**

```
get_svd(m_pmi, embedding_dim = 100, svd_rank = embedding_dim * 2)
```

**Arguments**

<code>m_pmi</code>	Pointwise mutual information matrix.
<code>embedding_dim</code>	Number of output embedding dimensions requested.
<code>svd_rank</code>	Number of SVD dimensions to compute.

**Value**

SVD rectangular matrix

**Examples**

```
df_ehr = data.frame(Patient = c(1, 1, 2, 1, 2, 1, 1, 3, 4),
                    Month = c(1, 1, 1, 2, 2, 3, 3, 4, 4),
                    Parent_Code = c('C1', 'C2', 'C2', 'C1', 'C1', 'C1',
                                     'C2', 'C3', 'C4'),
                    Count = 1:9)

spm_cooc = build_df_cooc(df_ehr)

m_pmi = get_pmi(spm_cooc)
m_svd = get_svd(m_pmi, embedding_dim = 2)
```

---

spm_to_df	<i>Write sparse matrix to dataframe</i>
-----------	---

---

**Description**

Write sparse matrix to dataframe

**Usage**

```
spm_to_df(spm)
```

**Arguments**

spm	Sparse matrix
-----	---------------

**Value**

Data frame

---

sql_cooc	<i>Compute co-occurrence matrix on SQL file</i>
----------	---

---

**Description**

Performs out-of-memory co-occurrence for large databases that would not fit in RAM memory with the classic call to `build_df_cooc`. Patients are batched using the `n_batch` parameter. Co-occurrence sparse matrix output is written to a new SQL file. Depending on number of codes considered, need to adjust `n_batch` and `n_cores`. See vignette "Co-occurrence and PMI-SVD" for more details.

**Usage**

```

sql_cooc(
  input_path,
  output_path,
  min_code_freq = 5,
  exclude_code_pattern = NULL,
  exclude_dict_pattern = NULL,
  codes_dict_fpaths = NULL,
  n_batch = 300,
  n_cores = 1,
  autoindex = FALSE,
  overwrite_output = FALSE,
  verbose = TRUE,
  verbose_max = verbose,
  ...
)

```

**Arguments**

<code>input_path</code>	Input SQL file path. Must contain monthly counts table 'df_monthly', with columns 'Patient', 'Month', 'Parent_Code', 'Count'. Also requires an index on column 'Patient' and a table of the unique codes 'df_uniq_codes', but will perform it automatically if parameter autoindex is TRUE (can increase input file size by 40%).
<code>output_path</code>	Output SQL file path for co-occurrence sparse matrix. Can overwrite with <code>overwrite_output</code> parameter.
<code>min_code_freq</code>	Filter output matrix based on code frequency.
<code>exclude_code_pattern</code>	Pattern of codes prefixes to exclude. Will be used in SQL appended by ' prefixed by '^'. For example, 'AB'.
<code>exclude_dict_pattern</code>	Used in combination with <code>codes_dict</code> . Pattern of codes prefixes to exclude, except if they are found in <code>codes_dict</code> . Will be used in SQL appended by ' grep prefixed by '^'. For example, 'C[0-9]'.
<code>codes_dict_fpaths</code>	Used in combination with <code>exclude_dict_pattern</code> . Filepaths to define codes to avoid excluding using <code>exclude_dict_pattern</code> . First column of each file must define the code identifiers.
<code>n_batch</code>	Number of patients per batch.
<code>n_cores</code>	Number of cores.
<code>autoindex</code>	If table 'df_uniq_codes' not found in <code>input_path</code> , index table 'df_monthly' on column 'Patient', and write unique values of 'Parent_Code' to table 'df_uniq_codes'.
<code>overwrite_output</code>	Should <code>output_path</code> be overwritten ?
<code>verbose</code>	Prints batch progress.

verbose\_max      Prints memory usage at each batch.  
 ...              Passed to build\_df\_cooc

**Value**

None, side-effect is output SQL file creation.

**Examples**

```
df_ehr = data.frame(Patient = c(1, 1, 2, 1, 2, 1, 1, 3, 4),
                    Month = c(1, 1, 1, 2, 2, 3, 3, 4, 4),
                    Parent_Code = c('C1', 'C2', 'C2', 'C1', 'C1', 'C1',
                                     'C2', 'C3', 'C4'),
                    Count = 1:9)

library(RSQLite)

test_db_path = tempfile()
test_db = dbConnect(SQLite(), test_db_path)
dbWriteTable(test_db, 'df_monthly', df_ehr, overwrite = TRUE)

dbDisconnect(test_db)

output_db_path = tempfile()
sql_cooc(test_db_path, output_db_path, autoindex = TRUE)

test_db = dbConnect(SQLite(), output_db_path)
spm_cooc = dbGetQuery(test_db, 'select * from df_monthly;')
dbDisconnect(test_db)

spm_cooc
```

**Description**

Pipe an object forward into a function or call expression and update the 'lhs' object with the resulting value. Magrittr imported function, see details and examples in the magrittr package.

**Arguments**

lhs              An object which serves both as the initial value and as target.  
 rhs              a function call using the magrittr semantics.

**Value**

None, used to update the value of lhs.

---

%% *Exposition pipe*

---

**Description**

Expose the names in 'lhs' to the 'rhs' expression. Magrittr imported function, see details and examples in the magrittr package.

**Arguments**

lhs            A list, environment, or a data.frame.  
rhs            An expression where the names in lhs is available.

**Value**

Result of rhs applied to one or several names of lhs.

---

%>% *Pipe*

---

**Description**

Pipe an object forward into a function or call expression. Magrittr imported function, see details and examples in the magrittr package.

**Arguments**

lhs            A value or the magrittr placeholder.  
rhs            A function call using the magrittr semantics.

**Value**

Result of rhs applied to lhs, see details in magrittr package.



# Index

[%<>%](#), 7

[%>%](#), 8

[%\\$%](#), 8

[build\\_df\\_cooc](#), 2

[build\\_spm\\_cooc\\_sym](#), 3

[get\\_pmi](#), 3

[get\\_svd](#), 4

[spm\\_to\\_df](#), 5

[sql\\_cooc](#), 5