

# Package: alethiaR (via r-universe)

October 16, 2026

**Title** Match and Deduplicate Messy Entity Names Against a Reference List

**Version** 0.1.0

**Description** Aligns messy entity strings against a reference list using either string similarity or language-model embeddings.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Depends** R (>= 4.1.0)

**Imports** stats, stringdist (>= 0.9.10)

**Suggests** testthat (>= 3.0.0), jsonlite, igraph, onnxr, tok, hfhub, knitr, rmarkdown

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**RoxygenNote** 8.0.0

**URL** <https://alethia.saketlab.org/cli/R>,  
<https://github.com/saketlab/alethia>

**BugReports** <https://github.com/saketlab/alethia/issues>

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

**Date/Publication** 2026-10-03 23:00:57 UTC

**RemoteUrl** <https://github.com/saketlab/alethia>

**RemoteRef** main

**RemoteSha** 7d3f46a4b4918a2fc8d1d1284a7ee60c20e9acd1

**RemoteSubdir** R

## Contents

|                          |   |
|--------------------------|---|
| alethia . . . . .        | 2 |
| alethia_split . . . . .  | 4 |
| alignment_loss . . . . . | 5 |

|                                    |    |
|------------------------------------|----|
| apply_threshold . . . . .          | 6  |
| as_embedder . . . . .              | 6  |
| assess-metrics . . . . .           | 7  |
| assess_models . . . . .            | 7  |
| centered_separability . . . . .    | 9  |
| cluster_entities . . . . .         | 10 |
| cluster_records . . . . .          | 11 |
| DEFAULT_THRESHOLD . . . . .        | 11 |
| encode . . . . .                   | 12 |
| hubness . . . . .                  | 12 |
| intrinsic_dimensionality . . . . . | 13 |
| l2_normalize . . . . .             | 14 |
| match_by_embeddings . . . . .      | 14 |
| match_by_strings . . . . .         | 15 |
| mutual_nn_edges . . . . .          | 16 |
| mutual_nn_rate . . . . .           | 17 |
| n_clusters . . . . .               | 18 |
| onnx_available . . . . .           | 18 |
| onnx_embedder . . . . .            | 19 |
| positive_pair_rank . . . . .       | 20 |
| RANK_DECIMALS . . . . .            | 21 |
| reference_separability . . . . .   | 21 |
| retrieval_margin . . . . .         | 22 |
| set_embedding_backend . . . . .    | 23 |
| token_sort . . . . .               | 24 |
| token_sort_ratio . . . . .         | 24 |
| true_accuracy . . . . .            | 25 |
| uniformity_loss . . . . .          | 26 |
| use_onnx_backend . . . . .         | 27 |

|              |           |
|--------------|-----------|
| <b>Index</b> | <b>28</b> |
|--------------|-----------|

---

|         |                                                            |
|---------|------------------------------------------------------------|
| alethia | <i>Align messy entity strings against a reference list</i> |
|---------|------------------------------------------------------------|

---

## Description

The main entry point, mirroring the Python ‘alethia()’. The pipeline is:

## Usage

```
alethia(
    dirty_entries,
    reference_entries,
    model = "rapidfuzz",
    threshold = NULL,
    use_exact_matching = TRUE,
    exact_match_case_sensitive = FALSE,
```

```

    drop_duplicates = TRUE
  )

```

## Arguments

**dirty\_entries** Character vector of messy strings.

**reference\_entries** Character vector of canonical strings.

**model** "rapidfuzz" for token-sort string similarity (the default, and the only backend needing no model), or anything [as\_embedder()] accepts.

**threshold** Minimum score to accept a match; below it the entry is reported as no match. 'NULL' means 'DEFAULT\_THRESHOLD' for embeddings and no threshold for string matching. Scores are on '[0, 1]' either way, but string similarity and cosine measure different things, so a value tuned against one is not calibrated for the other.

**use\_exact\_matching** Short-circuit entries already present in the references.

**exact\_match\_case\_sensitive** Whether that short-circuit respects case.

**drop\_duplicates** Collapse identical result rows, giving one row per distinct input. 'FALSE' keeps the input length and order, for joining back onto a table.

## Details

1. drop 'NA' and null-like entries, remembering their positions; 2. short-circuit entries that already appear in the reference list; 3. send whatever remains to the chosen backend; 4. splice all three groups back into the caller's original row order.

## Value

A data frame carrying 'given\_entity', 'alethia\_prediction', 'alethia\_score' and 'alethia\_method', in input order.

## Examples

```

alethia(
  c("Bombay", "Calcutta", "Mumbai"),
  c("Mumbai", "Kolkata", "Chennai")
)

# Entries already present in the references short-circuit as exact matches, and
# null-like values are preserved as no-match rows rather than dropped.
alethia(c("Mumbai", NA, "N/A"), c("Mumbai"), drop_duplicates = FALSE)

```

alethia\_split

*Match composite entries by splitting them on a separator first***Description**

Mirrors the Python ‘alethia\_split()’. Each entry is split into pieces on a literal separator, every piece is matched independently against ‘reference\_entries’ via [alethia()], and the highest-scoring piece becomes the entry’s prediction. Ties break on first occurrence, same as the rest of the package. Every piece’s own match rides along in ‘alethia\_parts’.

**Usage**

```
alethia_split(dirty_entries, reference_entries, split_on, ...)
```

**Arguments**

`dirty_entries`    Character vector of messy strings.

`reference_entries`  
                     Character vector of canonical strings.

`split_on`         Literal separator each entry is split on (e.g. “,”, “;”, “.” and “”). Entries with no separator, or that split to nothing usable, are matched whole.

`...`             Forwarded to [alethia()] for the per-piece matching (‘model’, ‘threshold’, ...).

**Value**

A data frame of ‘given\_entity’, ‘alethia\_prediction’, ‘alethia\_score’, ‘alethia\_matched\_part’ (the winning piece), and ‘alethia\_parts’ (a list column: every piece tried, as a list of ‘part’/‘prediction’/‘score’, best first).

**Examples**

```
alethia_split(
  c("CNS Tuberculosis, Septic Shock"),
  c("Tuberculosis of nervous system", "Septic shock"),
  split_on = ", "
)
```

---

|                |                       |
|----------------|-----------------------|
| alignment_loss | <i>Alignment loss</i> |
|----------------|-----------------------|

---

### Description

Mean  $\|x - y\|^\alpha$  over matched positive pairs, on L2-normalised embeddings. Lower means the model keeps a string and its dirtied variant close. After Wang & Isola (2020).

### Usage

```
alignment_loss(src_emb, var_emb, alpha = 2)
```

### Arguments

|                  |                                                      |
|------------------|------------------------------------------------------|
| src_emb, var_emb | Matched embedding matrices, row 'i' a positive pair. |
| alpha            | Exponent on the distance.                            |

### Value

Scalar loss.

### See Also

Other assess metrics: [centered\\_separability\(\)](#), [hubness\(\)](#), [intrinsic\\_dimensionality\(\)](#), [mutual\\_nn\\_rate\(\)](#), [positive\\_pair\\_rank\(\)](#), [reference\\_separability\(\)](#), [retrieval\\_margin\(\)](#), [uniformity\\_loss\(\)](#)

### Examples

```
bag <- function(x) {
  t(sapply(
    strsplit(tolower(x), ""),
    function(ch) tabulate(match(ch, letters), 26)
  ))
}
refs <- bag(c("aspirin", "ibuprofen", "paracetamol", "metformin"))
queries <- bag(c("asprin", "ibuprofn", "paracetmol", "metfornin"))
alignment_loss(refs, queries)
```

---

|                 |                                                    |
|-----------------|----------------------------------------------------|
| apply_threshold | <i>Blank out matches scoring below a threshold</i> |
|-----------------|----------------------------------------------------|

---

### Description

Sub-threshold rows are reported as no match. ‘NULL’ leaves the frame untouched.

### Usage

```
apply_threshold(results, threshold)
```

### Arguments

|           |                                                           |
|-----------|-----------------------------------------------------------|
| results   | Data frame with ‘alethia_prediction’ and ‘alethia_score’. |
| threshold | Minimum acceptable score, or ‘NULL’.                      |

### Value

The frame, with sub-threshold rows set to ‘NA’.

### Examples

```
res <- match_by_strings("Bombay", c("Mumbai", "Kolkata"))
apply_threshold(res, 0.9)
```

---

|             |                                                      |
|-------------|------------------------------------------------------|
| as_embedder | <i>Coerce a model specification into an embedder</i> |
|-------------|------------------------------------------------------|

---

### Description

Mirrors ‘alethia.embedder.as\_embedder’. An embedder is any object with an ‘encode(texts)’ method returning an ‘n x d’ numeric matrix. This function accepts:

### Usage

```
as_embedder(model, name = NULL)
```

### Arguments

|       |                                           |
|-------|-------------------------------------------|
| model | A function, an embedder, or a model name. |
| name  | Optional display name.                    |

## Details

\* a function 'f(texts) -> matrix', wrapped as a callable embedder; \* an object already carrying an 'encode' element (returned unchanged); \* a character model name, resolved through the registered backend (see [set\_embedding\_backend()]).

Every embedding source sits behind this seam, so matching, clustering and assessment share one implementation.

## Value

An object of class 'alethia\_embedder'.

## Examples

```
# Any function from character vector to matrix is an embedder.
bag <- function(x) {
  t(sapply(
    strsplit(tolower(x), ""),
    function(ch) tabulate(match(ch, letters), 26)
  ))
}
e <- as_embedder(bag, name = "letter-count")
e
```

---

|                |                                                          |
|----------------|----------------------------------------------------------|
| assess-metrics | <i>Label-free geometric metrics for an embedding set</i> |
|----------------|----------------------------------------------------------|

---

## Description

These score an embedding model on your own data without ground-truth labels, by measuring the geometry it produces. Ports of 'alethia.assess.metrics'; each formula is stated so the two implementations can be checked against one another.

---

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| assess_models | <i>Score candidate embedding models on your own data, without labels</i> |
|---------------|--------------------------------------------------------------------------|

---

## Description

Each model is scored on the metrics in [assess-metrics], z-scored **across models** and summed. The composite therefore needs at least two models; with one the score is 'NA' and only the component metrics are reported. Weights are normalised **within metric family**, so correlated views of one property cannot count it twice, and a family that could not be computed drops out.

**Usage**

```

assess_models(
  queries,
  references,
  models,
  weights = NULL,
  family_normalize = TRUE,
  variants = NULL
)

DEFAULT_METRIC_WEIGHTS

```

**Arguments**

|                               |                                                                                                                                                                                                                                                                                   |
|-------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>queries</code>          | Character vector of dirty/query strings. May be empty, in which case the retrieval family is skipped and the reference geometry alone is scored.                                                                                                                                  |
| <code>references</code>       | Character vector of canonical strings.                                                                                                                                                                                                                                            |
| <code>models</code>           | Named list of model specifications, each anything <code>[as_embedder()]</code> accepts. Names are used for reporting.                                                                                                                                                             |
| <code>weights</code>          | Optional named numeric vector overriding the per-metric weights. Interpreted within family when <code>'family_normalize'</code> is <code>'TRUE'</code> .                                                                                                                          |
| <code>family_normalize</code> | Rescale weights so each family contributes at most its family weight. <code>'FALSE'</code> gives flat per-metric weighting.                                                                                                                                                       |
| <code>variants</code>         | Optional list of <code>'source'</code> and <code>'variant'</code> character vectors, row <code>'i'</code> a (clean, dirtied) pair, which enables the robustness family. Generate them with <code>'alethia.assess.generate_positive_pairs'</code> ; omitted, the family drops out. |

**Value**

An object of class `'alethia_assessment'` with `'$table'` (a data frame, best first) and `'$best'` (the winning model name, or `'NA'` when not comparative).

**Examples**

```

bag <- function(x) {
  t(sapply(
    strsplit(tolower(x), ""),
    function(ch) tabulate(match(ch, letters), 26)
  ))
}
rev_bag <- function(texts) {
  bag(vapply(texts, function(s) {
    paste(rev(strsplit(s, "")[[1]]), collapse = "")
  }, character(1)))
}
refs <- c("aspirin", "ibuprofen", "paracetamol", "metformin", "atenolol")
res <- assess_models(

```

```

      c("asprin", "ibuprofn"), refs,
      list(bag = bag, reversed = rev_bag)
    )
  res$best

```

---

centered\_separability *Anisotropy-corrected separability*

---

## Description

[reference\_separability()] after removing the set mean, which strips the shared direction responsible for uniformly high cosines.

## Usage

```
centered_separability(ref_emb)
```

## Arguments

ref\_emb            Numeric matrix of reference embeddings.

## Value

Named list of ‘centered\_nn\_similarity’ and ‘nn\_margin\_z’.

## See Also

Other assess metrics: [alignment\\_loss\(\)](#), [hubness\(\)](#), [intrinsic\\_dimensionality\(\)](#), [mutual\\_nn\\_rate\(\)](#), [positive\\_pair\\_rank\(\)](#), [reference\\_separability\(\)](#), [retrieval\\_margin\(\)](#), [uniformity\\_loss\(\)](#)

## Examples

```

bag <- function(x) {
  t(sapply(
    strsplit(tolower(x), ""),
    function(ch) tabulate(match(ch, letters), 26)
  ))
}
refs <- bag(c("aspirin", "ibuprofen", "paracetamol", "metformin"))
queries <- bag(c("asprin", "ibuprofn", "paracetmol", "metfornin"))
centered_separability(refs)

```

---

|                  |                                                         |
|------------------|---------------------------------------------------------|
| cluster_entities | <i>Group entity strings into clusters of duplicates</i> |
|------------------|---------------------------------------------------------|

---

## Description

Mirrors ‘alethia.cluster.cluster\_entities’. Edges require *\*mutual\** nearest-neighbourhood above a cosine floor and each carries a confidence, so no hub chains the whole set into one cluster the way a plain threshold does.

## Usage

```
cluster_entities(
  entities,
  model,
  floor = 0.8,
  k = 5,
  require_mutual = TRUE,
  min_confidence = NULL,
  canonical = c("shortest", "first")
)
```

## Arguments

|                |                                                                     |
|----------------|---------------------------------------------------------------------|
| entities       | Character vector to group. De-duplicated on exact text first.       |
| model          | Anything [as_embedder()] accepts.                                   |
| floor          | Minimum cosine for an edge.                                         |
| k              | Neighbourhood size for the mutual-NN test. Clamped to ‘[1, n - 1]’. |
| require_mutual | Require each entity to be in the other’s top-k.                     |
| min_confidence | Optional cutoff applied to edge confidence before clustering.       |
| canonical      | How to name each cluster: “shortest” member or “first” seen.        |

## Value

An object of class ‘alethia\_clusters’.

## Examples

```
bag <- function(x) {
  t(sapply(
    strsplit(tolower(x), ""),
    function(ch) tabulate(match(ch, letters), 26)
  ))
}
res <- cluster_entities(c("color", "colour", "zebra"), bag, floor = 0.8)
res
cluster_records(res)
```

---

|                 |                                                |
|-----------------|------------------------------------------------|
| cluster_records | <i>Cluster membership as a tidy data frame</i> |
|-----------------|------------------------------------------------|

---

**Description**

Cluster membership as a tidy data frame

**Usage**

```
cluster_records(x)
```

**Arguments**

`x` An ‘alethia\_clusters’ object.

**Value**

A data frame with ‘entity’, ‘cluster’ and ‘canonical’ columns.

**Examples**

```
bag <- function(x) {
  t(sapply(
    strsplit(tolower(x), ""),
    function(ch) tabulate(match(ch, letters), 26)
  ))
}
cluster_records(cluster_entities(c("color", "colour"), bag, floor = 0.8))
```

---

|                   |                                                                 |
|-------------------|-----------------------------------------------------------------|
| DEFAULT_THRESHOLD | <i>Default minimum cosine similarity for embedding backends</i> |
|-------------------|-----------------------------------------------------------------|

---

**Description**

String matching does not use this; a cosine and a surface-overlap score of the same value do not mean the same thing.

**Usage**

```
DEFAULT_THRESHOLD
```

---

|        |                                     |
|--------|-------------------------------------|
| encode | <i>Encode text with an embedder</i> |
|--------|-------------------------------------|

---

**Description**

Encode text with an embedder

**Usage**

```
encode(embedder, texts)
```

**Arguments**

|          |                                              |
|----------|----------------------------------------------|
| embedder | An 'alethia_embedder', from [as_embedder()]. |
| texts    | Character vector.                            |

**Value**

Numeric matrix with one row per input.

**Examples**

```
bag <- function(x) {
  t(sapply(
    strsplit(tolower(x), ""),
    function(ch) tabulate(match(ch, letters), 26)
  ))
}
dim(encode(as_embedder(bag), c("colour", "color")))
```

---

|         |                |
|---------|----------------|
| hubness | <i>Hubness</i> |
|---------|----------------|

---

**Description**

Skewness of the k-occurrence distribution: how unevenly references are drawn as top-k neighbours. High values are pathological, a few "hub" references absorbing many queries. After Radovanovic et al. (2010), JMLR 11:2487-2531.

**Usage**

```
hubness(query_emb, ref_emb, k = 5)
```

**Arguments**

|                    |                             |
|--------------------|-----------------------------|
| query_emb, ref_emb | Numeric embedding matrices. |
| k                  | Neighbourhood size.         |

**Value**

Named list with 'hubness\_skew'.

**See Also**

Other assess metrics: [alignment\\_loss\(\)](#), [centered\\_separability\(\)](#), [intrinsic\\_dimensionality\(\)](#), [mutual\\_nn\\_rate\(\)](#), [positive\\_pair\\_rank\(\)](#), [reference\\_separability\(\)](#), [retrieval\\_margin\(\)](#), [uniformity\\_loss\(\)](#)

**Examples**

```
bag <- function(x) {
  t(sapply(
    strsplit(tolower(x), ""),
    function(ch) tabulate(match(ch, letters), 26)
  ))
}
refs <- bag(c("aspirin", "ibuprofen", "paracetamol", "metformin"))
queries <- bag(c("asprin", "ibuprofn", "paracetmol", "metfornin"))
hubness(queries, refs, k = 2)
```

---

intrinsic\_dimensionality

*Intrinsic dimensionality*


---

**Description**

Participation ratio  $(\sum(\lambda)^2 / \sum(\lambda^2))$  over the PCA eigenvalues, and the same divided by the ambient dimension. Higher means the model spreads the data over more directions.

**Usage**

```
intrinsic_dimensionality(emb)
```

**Arguments**

emb                      Numeric matrix of embeddings.

**Value**

Named list of 'participation\_ratio' and 'normalized\_pr'.

**See Also**

Other assess metrics: [alignment\\_loss\(\)](#), [centered\\_separability\(\)](#), [hubness\(\)](#), [mutual\\_nn\\_rate\(\)](#), [positive\\_pair\\_rank\(\)](#), [reference\\_separability\(\)](#), [retrieval\\_margin\(\)](#), [uniformity\\_loss\(\)](#)

**Examples**

```

bag <- function(x) {
  t(sapply(
    strsplit(tolower(x), ""),
    function(ch) tabulate(match(ch, letters), 26)
  ))
}
refs <- bag(c("aspirin", "ibuprofen", "paracetamol", "metformin"))
queries <- bag(c("asprin", "ibuprofn", "paracetmol", "metfornin"))
intrinsic_dimensionality(refs)

```

---

l2\_normalize*Row-wise L2 normalisation*

---

**Description**

The single canonical normalisation used by matching, clustering and assessment, mirroring ‘alethia.embedder.l2\_normalize’. Zero rows stay zero, never producing ‘NaN’.

**Usage**

```
l2_normalize(x)
```

**Arguments**

**x** Numeric matrix, one embedding per row.

**Value**

Matrix of the same shape with unit-norm rows.

**Examples**

```
l2_normalize(rbind(c(3, 4), c(0, 0)))
```

---

match\_by\_embeddings*Match queries to references by embedding cosine similarity*

---

**Description**

The single embedding-based matching primitive, mirroring ‘alethia.embedder.match\_by\_embeddings’. Every embedding backend routes through it.

**Usage**

```
match_by_embeddings(queries, references, embedder, threshold = NULL)
```

**Arguments**

|            |                                                        |
|------------|--------------------------------------------------------|
| queries    | Character vector of strings to match.                  |
| references | Character vector of candidate strings.                 |
| embedder   | Anything [as_embedder()] accepts.                      |
| threshold  | Optional minimum cosine. Matches below it become 'NA'. |

**Value**

A data frame with 'given\_entity', 'alethia\_prediction', 'alethia\_score'.

**Examples**

```
bag <- function(x) {
  t(sapply(
    strsplit(tolower(x), ""),
    function(ch) tabulate(match(ch, letters), 26)
  ))
}
match_by_embeddings(c("colour"), c("color", "zebra"), bag)
```

---

|                  |                                                                        |
|------------------|------------------------------------------------------------------------|
| match_by_strings | <i>Match strings against a reference list by token-sort similarity</i> |
|------------------|------------------------------------------------------------------------|

---

**Description**

The counterpart of Python's 'run\_rapidfuzz\_matching()': for each query, the highest-scoring reference.

**Usage**

```
match_by_strings(queries, references, threshold = NULL)
```

**Arguments**

|            |                                                                                                         |
|------------|---------------------------------------------------------------------------------------------------------|
| queries    | Character vector of strings to match.                                                                   |
| references | Character vector of candidate strings.                                                                  |
| threshold  | Minimum score in '[0, 1]'; below it an entry is 'NA'. 'NULL' returns the best match whatever it scores. |

**Value**

A data frame with 'given\_entity', 'alethia\_prediction' and 'alethia\_score'. Scores are rescaled to '[0, 1]'.

**Examples**

```

match_by_strings(
  c("Bombay", "Calcutta"),
  c("Mumbai", "Kolkata", "Chennai")
)

# Demand a close match, and report anything weaker as no match.
match_by_strings("Bombay", c("Mumbai", "Kolkata"), threshold = 0.9)

```

---

|                 |                                                             |
|-----------------|-------------------------------------------------------------|
| mutual_nn_edges | <i>Candidate merge edges from mutual nearest neighbours</i> |
|-----------------|-------------------------------------------------------------|

---

**Description**

Candidate merge edges from mutual nearest neighbours

**Usage**

```
mutual_nn_edges(emb, floor = 0.8, k = 5, require_mutual = TRUE)
```

**Arguments**

|                |                                                                  |
|----------------|------------------------------------------------------------------|
| emb            | Numeric matrix, one embedding per row. L2-normalised internally. |
| floor          | Minimum cosine similarity for an edge.                           |
| k              | Neighbourhood size.                                              |
| require_mutual | Keep only reciprocated edges.                                    |

**Value**

A data frame of edges with 'i', 'j', 'cosine', 'margin\_i', 'margin\_j', 'mutual' and 'confidence', sorted by decreasing confidence.

**Examples**

```

emb <- rbind(c(1, 0), c(0.99, 0.14), c(0, 1))
mutual_nn_edges(emb, floor = 0.5, k = 1)

```

---

|                |                                      |
|----------------|--------------------------------------|
| mutual_nn_rate | <i>Mutual nearest-neighbour rate</i> |
|----------------|--------------------------------------|

---

## Description

Fraction of query-to-reference top-1 assignments reciprocated within the reference's own top-k. Higher means more stable, more trustworthy matches.

## Usage

```
mutual_nn_rate(query_emb, ref_emb, k = 5)
```

## Arguments

|                    |                                           |
|--------------------|-------------------------------------------|
| query_emb, ref_emb | Numeric embedding matrices.               |
| k                  | Neighbourhood size on the reference side. |

## Value

Scalar rate.

## See Also

Other assess metrics: [alignment\\_loss\(\)](#), [centered\\_separability\(\)](#), [hubness\(\)](#), [intrinsic\\_dimensionality\(\)](#), [positive\\_pair\\_rank\(\)](#), [reference\\_separability\(\)](#), [retrieval\\_margin\(\)](#), [uniformity\\_loss\(\)](#)

## Examples

```
bag <- function(x) {  
  t(sapply(  
    strsplit(tolower(x), ""),  
    function(ch) tabulate(match(ch, letters), 26)  
  ))  
}  
refs <- bag(c("aspirin", "ibuprofen", "paracetamol", "metformin"))  
queries <- bag(c("asprin", "ibuprofn", "paracetmol", "metfornin"))  
mutual_nn_rate(queries, refs, k = 2)
```

---

|            |                                 |
|------------|---------------------------------|
| n_clusters | <i>Number of clusters found</i> |
|------------|---------------------------------|

---

**Description**

Number of clusters found

**Usage**

```
n_clusters(x)
```

**Arguments**

x                      An ‘alethia\_clusters’ object.

**Value**

Integer count of distinct clusters.

**Examples**

```
bag <- function(x) {
  t(sapply(
    strsplit(tolower(x), ""),
    function(ch) tabulate(match(ch, letters), 26)
  ))
}
n_clusters(cluster_entities(c("color", "colour", "zebra"), bag, floor = 0.8))
```

---

|                |                                              |
|----------------|----------------------------------------------|
| onnx_available | <i>Is the local ONNX stack ready to use?</i> |
|----------------|----------------------------------------------|

---

**Description**

Checks that the three R packages are installed and that ONNX Runtime itself has been downloaded. The runtime is a separate one-time download, not an R dependency, so installing the packages is not enough.

**Usage**

```
onnx_available()
```

**Value**

‘TRUE’ when local ONNX inference will work.

**Examples**

```
onnx_available()
```

onnx\_embedder

*Build an embedder backed by a local ONNX model***Description**

Runs a sentence-embedding model on the CPU with no Python installation, using 'onnxr' for inference, 'tok' for tokenisation, and 'hfhub' to fetch the weights.

**Usage**

```
onnx_embedder(
    model_id,
    backend = c("cpu", "coreml", "cuda", "xnnpack", "openvino"),
    threads = NULL,
    max_length = 256L,
    batch_size = 32L,
    onnx_file = "onnx/model.onnx",
    tokenizer_file = "tokenizer.json"
)
```

**Arguments**

|                |                                                                                                                                      |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------|
| model_id       | Hugging Face model id, e.g. "sentence-transformers/all-MiniLM-L6-v2".                                                                |
| backend        | Execution provider. "cpu" by default; the others are only useful if you have the matching hardware and an ONNX Runtime built for it. |
| threads        | Inference threads. 'NULL' leaves it to ONNX Runtime.                                                                                 |
| max_length     | Sequence length above which inputs are truncated.                                                                                    |
| batch_size     | Texts per forward pass. Lower this if memory is tight; it changes speed and memory, not results.                                     |
| onnx_file      | Path within the repository to the ONNX weights.                                                                                      |
| tokenizer_file | Path within the repository to the tokeniser.                                                                                         |

**Details**

Pooling reproduces sentence-transformers (an attention-masked mean over token vectors, then L2 normalisation), so embeddings agree with the Python package to floating-point tolerance for the same model.

The model must publish ONNX weights. Most sentence-transformers models do, under 'onnx/model.onnx'; the 'onnx-community' and 'Xenova' organisations on Hugging Face publish conversions of many others.

**Value**

An 'alethia\_embedder', usable anywhere a model is accepted.

**See Also**

[`use_onnx_backend()`] to resolve bare model names through this path.

**Examples**

```
# Downloads ~90 MB on first use, then caches.
if (onnx_available()) {
  embedder <- onnx_embedder("sentence-transformers/all-MiniLM-L6-v2")
  alethia(c("Bombay", "Calcutta"), c("Mumbai", "Kolkata"), model = embedder)
}
```

---

|                    |                           |
|--------------------|---------------------------|
| positive_pair_rank | <i>Positive-pair rank</i> |
|--------------------|---------------------------|

---

**Description**

How highly each synthesized positive ranks among *\*all\** variants. ‘src\_emb[i]’ and ‘var\_emb[i]’ are a positive pair; for each source this is the percentile rank of its own variant among every variant, averaged. Higher is better. Ties take their mid-rank.

**Usage**

```
positive_pair_rank(src_emb, var_emb)
```

**Arguments**

```
src_emb, var_emb
```

Matched embedding matrices, row ‘i’ a positive pair.

**Details**

Unlike [`alignment_loss()`], this uses only the *\*ordering\** of each source’s similarities, so it is invariant to any strictly increasing transformation of cosine, including the affine shift from adding a common cone direction and renormalising. Hence its weight in the composite.

**Value**

Scalar mean percentile rank in ‘[0, 1]’.

**See Also**

Other assess metrics: [alignment\\_loss\(\)](#), [centered\\_separability\(\)](#), [hubness\(\)](#), [intrinsic\\_dimensionality\(\)](#), [mutual\\_nn\\_rate\(\)](#), [reference\\_separability\(\)](#), [retrieval\\_margin\(\)](#), [uniformity\\_loss\(\)](#)

---

|               |                                                                               |
|---------------|-------------------------------------------------------------------------------|
| RANK_DECIMALS | <i>Decimal places similarities are rounded to before any ranking decision</i> |
|---------------|-------------------------------------------------------------------------------|

---

**Description**

Equally similar references are common in real data: shared tokens, anagrams, duplicated records. Whether such a pair is *\*seen\** as tied otherwise depends on the last bits of a dot product, which differ between BLAS builds and languages. Rounding first makes "equally similar" mean the same thing everywhere, so the tie-break by ascending index gives the same answer. Reported scores are left unrounded.

**Usage**

RANK\_DECIMALS

---

|                        |                                   |
|------------------------|-----------------------------------|
| reference_separability | <i>Reference-set separability</i> |
|------------------------|-----------------------------------|

---

**Description**

How well distinct references are kept apart. Returns the mean and median cosine to the nearest *\*other\** reference (lower is better), and the fraction whose nearest other reference exceeds *'confusion\_threshold'*.

**Usage**

reference\_separability(ref\_emb, confusion\_threshold = 0.9)

**Arguments**

- ref\_emb                      Numeric matrix of reference embeddings.
- confusion\_threshold                      Cosine above which two references count as confusable.

**Details**

These are raw cosines, so they partly reflect anisotropy: transformer embeddings occupy a narrow cone, pushing every cosine high. Retrieval depends on the ordering of similarities, not their level, so [centered\_separability()] gives the anisotropy-corrected view.

**Value**

Named list of *'mean\_nn\_similarity'*, *'median\_nn\_similarity'*, *'confusability\_rate'*.

**See Also**

Other assess metrics: [alignment\\_loss\(\)](#), [centered\\_separability\(\)](#), [hubness\(\)](#), [intrinsic\\_dimensionality\(\)](#), [mutual\\_nn\\_rate\(\)](#), [positive\\_pair\\_rank\(\)](#), [retrieval\\_margin\(\)](#), [uniformity\\_loss\(\)](#)

**Examples**

```
bag <- function(x) {
  t(sapply(
    strsplit(tolower(x), ""),
    function(ch) tabulate(match(ch, letters), 26)
  ))
}
refs <- bag(c("aspirin", "ibuprofen", "paracetamol", "metformin"))
queries <- bag(c("asprin", "ibuprofn", "paracetmol", "metfornin"))
reference_separability(refs)
```

---

retrieval\_margin

*Retrieval margin*


---

**Description**

How decisively the top reference beats the runner-up for each query, both in raw cosine and in per-query standard deviations (`'_z'`, which is scale-free and so comparable across models with different cosine ranges).

**Usage**

```
retrieval_margin(query_emb, ref_emb, low_margin = 0.02)
```

**Arguments**

query\_emb, ref\_emb

Numeric embedding matrices.

low\_margin

Margin below which a match counts as ambiguous.

**Value**

Named list of `'mean_margin'`, `'low_margin_rate'`, `'mean_margin_z'`, `'low_margin_z_rate'`.

**See Also**

Other assess metrics: [alignment\\_loss\(\)](#), [centered\\_separability\(\)](#), [hubness\(\)](#), [intrinsic\\_dimensionality\(\)](#), [mutual\\_nn\\_rate\(\)](#), [positive\\_pair\\_rank\(\)](#), [reference\\_separability\(\)](#), [uniformity\\_loss\(\)](#)

**Examples**

```
bag <- function(x) {
  t(sapply(
    strsplit(tolower(x), ""),
    function(ch) tabulate(match(ch, letters), 26)
  ))
}
refs <- bag(c("aspirin", "ibuprofen", "paracetamol", "metformin"))
queries <- bag(c("asprin", "ibuprofn", "paracetmol", "metfornin"))
retrieval_margin(queries, refs)
```

---

set\_embedding\_backend *Register a backend that turns model names into embedding functions*

---

**Description**

The package ships no model runtime of its own. Point this at whatever you use to produce embeddings, e.g. a local ONNX session, an API client, or a cached matrix.

**Usage**

```
set_embedding_backend(backend)

get_embedding_backend()
```

**Arguments**

|         |                                                                                                               |
|---------|---------------------------------------------------------------------------------------------------------------|
| backend | A function taking a model name and returning a function ‘f(texts) -> matrix’. ‘NULL’ clears the registration. |
|---------|---------------------------------------------------------------------------------------------------------------|

**Value**

The previous backend, invisibly.

**Examples**

```
bag <- function(x) {
  t(sapply(
    strsplit(tolower(x), ""),
    function(ch) tabulate(match(ch, letters), 26)
  ))
}
old <- set_embedding_backend(function(model_name) bag)
as_embedder("any-model-name")
set_embedding_backend(old)
```

---

|            |                                                                      |
|------------|----------------------------------------------------------------------|
| token_sort | <i>Normalise a string by sorting its whitespace-separated tokens</i> |
|------------|----------------------------------------------------------------------|

---

**Description**

The preprocessing step inside [token\_sort\_ratio()], exported because the same normalisation is useful when caching or de-duplicating inputs.

**Usage**

```
token_sort(x)
```

**Arguments**

x                      Character vector.

**Value**

Character vector with tokens sorted and single-space separated.

**Examples**

```
token_sort(" South 24 Parganas ")
```

---

|                  |                                    |
|------------------|------------------------------------|
| token_sort_ratio | <i>Token-sort similarity ratio</i> |
|------------------|------------------------------------|

---

**Description**

Reproduces 'rapidfuzz.fuzz.token\_sort\_ratio', the scorer behind the default "rapidfuzz" backend.

**Usage**

```
token_sort_ratio(a, b)
```

**Arguments**

a, b                      Character vectors, recycled against each other.

**Details**

Exact agreement rests on an identity: RapidFuzz's 'ratio' is the normalised indel (insert/delete-only edit) similarity,

$$\text{ratio}(a, b) = 100 * (1 - \text{indel}(a, b) / (\text{nchar}(a) + \text{nchar}(b)))$$

and 'stringdist(method = "lcs")' computes that indel distance. Both sides evaluate the same rational number in double precision, so the scores agree to within an ULP.

Tokens are split on whitespace, sorted, and rejoined with a single space. Case and punctuation are left alone, so "bombay" and "Bombay" score 83.33, not 100.

**Value**

Numeric vector of scores in `'[0, 100]'`.

**Examples**

```
# Token order does not matter, but case does.
token_sort_ratio("New Delhi", "Delhi New")
token_sort_ratio("Bombay", "Mumbai")
```

---

|               |                                                                   |
|---------------|-------------------------------------------------------------------|
| true_accuracy | <i>Top-1 accuracy and MRR of an embedder against ground truth</i> |
|---------------|-------------------------------------------------------------------|

---

**Description**

The labelled counterpart to the label-free metrics: given the correct reference for each query, how often does the model retrieve it? `[assess_models()]` predicts this without labels.

**Usage**

```
true_accuracy(queries, references, truth, model, nil_threshold = NULL)
```

**Arguments**

|               |                                                                                                                                                                                                                     |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| queries       | Character vector of query strings.                                                                                                                                                                                  |
| references    | Character vector of candidate strings.                                                                                                                                                                              |
| truth         | Correct reference per query, or <code>'NA'</code> for an open-world NIL query with no correct answer. Real matching input is full of these: a cause of death that is not a diagnosis, a drug outside the formulary. |
| model         | Anything <code>[as_embedder()]</code> accepts.                                                                                                                                                                      |
| nil_threshold | Cosine below which a top-1 match counts as an abstention. Required when <code>'truth'</code> contains <code>'NA'</code> ; ignored otherwise.                                                                        |

**Value**

A named list with `'top1'` and `'mrr'`, plus `'nil_recall'` and `'answerable_top1'` when the dataset contains NIL queries. Those two are separate because one accuracy number hides the trade-off: a model can look good by always abstaining, or never.

**Examples**

```
bag <- function(x) {
  t(sapply(
    strsplit(tolower(x), ""),
    function(ch) tabulate(match(ch, letters), 26)
  ))
}
true_accuracy(
```

```
c("colour", "flavour"), c("color", "flavor"),
c("color", "flavor"), bag
)
```

---

|                 |                        |
|-----------------|------------------------|
| uniformity_loss | <i>Uniformity loss</i> |
|-----------------|------------------------|

---

## Description

‘ $\log E \exp(-t * \|x - y\|^2)$ ’ over all pairs. More negative means the embeddings spread more evenly over the sphere, wasting less of the representation. After Wang & Isola (2020).

## Usage

```
uniformity_loss(emb, t = 2, max_points = 2000)
```

## Arguments

|            |                                                    |
|------------|----------------------------------------------------|
| emb        | Numeric matrix of embeddings.                      |
| t          | Temperature.                                       |
| max_points | Subsample size when there are more rows than this. |

## Value

Scalar loss.

## See Also

Other assess metrics: [alignment\\_loss\(\)](#), [centered\\_separability\(\)](#), [hubness\(\)](#), [intrinsic\\_dimensionality\(\)](#), [mutual\\_nn\\_rate\(\)](#), [positive\\_pair\\_rank\(\)](#), [reference\\_separability\(\)](#), [retrieval\\_margin\(\)](#)

## Examples

```
bag <- function(x) {
  t(sapply(
    strsplit(tolower(x), ""),
    function(ch) tabulate(match(ch, letters), 26)
  ))
}
refs <- bag(c("aspirin", "ibuprofen", "paracetamol", "metformin"))
queries <- bag(c("asprin", "ibuprofn", "paracetmol", "metfornin"))
uniformity_loss(refs)
```

---

|                  |                                                                |
|------------------|----------------------------------------------------------------|
| use_onnx_backend | <i>Resolve bare model names through the local ONNX runtime</i> |
|------------------|----------------------------------------------------------------|

---

**Description**

After calling this, a model name passed to [alethia()], [cluster\_entities()] or [as\_embedder()] is loaded and run locally on the CPU.

**Usage**

```
use_onnx_backend(...)
```

**Arguments**

...                      Passed to [onnx\_embedder()], e.g. 'threads' or 'batch\_size'.

**Value**

The previously registered backend, invisibly.

**Examples**

```
if (onnx_available()) {  
  use_onnx_backend(threads = 4)  
  alethia(c("Bombay"), c("Mumbai", "Delhi"), model = "sentence-transformers/all-MiniLM-L6-v2")  
}
```

# Index

## \* assess metrics

- alignment\_loss, [5](#)
- centered\_separability, [9](#)
- hubness, [12](#)
- intrinsic\_dimensionality, [13](#)
- mutual\_nn\_rate, [17](#)
- positive\_pair\_rank, [20](#)
- reference\_separability, [21](#)
- retrieval\_margin, [22](#)
- uniformity\_loss, [26](#)

alethia, [2](#)

alethia\_split, [4](#)

alignment\_loss, [5](#)

alignment\_loss(), [9](#), [13](#), [17](#), [20](#), [22](#), [26](#)

apply\_threshold, [6](#)

as\_embedder, [6](#)

assess-metrics, [7](#)

assess\_models, [7](#)

  

centered\_separability, [9](#)

centered\_separability(), [5](#), [13](#), [17](#), [20](#), [22](#), [26](#)

cluster\_entities, [10](#)

cluster\_records, [11](#)

  

DEFAULT\_METRIC\_WEIGHTS (assess\_models), [7](#)

DEFAULT\_THRESHOLD, [11](#)

  

encode, [12](#)

  

get\_embedding\_backend  
(set\_embedding\_backend), [23](#)

  

hubness, [12](#)

hubness(), [5](#), [9](#), [13](#), [17](#), [20](#), [22](#), [26](#)

  

intrinsic\_dimensionality, [13](#)

intrinsic\_dimensionality(), [5](#), [9](#), [13](#), [17](#), [20](#), [22](#), [26](#)

l2\_normalize, [14](#)

  

match\_by\_embeddings, [14](#)

match\_by\_strings, [15](#)

mutual\_nn\_edges, [16](#)

mutual\_nn\_rate, [17](#)

mutual\_nn\_rate(), [5](#), [9](#), [13](#), [20](#), [22](#), [26](#)

  

n\_clusters, [18](#)

  

onnx\_available, [18](#)

onnx\_embedder, [19](#)

  

positive\_pair\_rank, [20](#)

positive\_pair\_rank(), [5](#), [9](#), [13](#), [17](#), [22](#), [26](#)

  

RANK\_DECIMALS, [21](#)

reference\_separability, [21](#)

reference\_separability(), [5](#), [9](#), [13](#), [17](#), [20](#), [22](#), [26](#)

retrieval\_margin, [22](#)

retrieval\_margin(), [5](#), [9](#), [13](#), [17](#), [20](#), [22](#), [26](#)

  

set\_embedding\_backend, [23](#)

  

token\_sort, [24](#)

token\_sort\_ratio, [24](#)

true\_accuracy, [25](#)

  

uniformity\_loss, [26](#)

uniformity\_loss(), [5](#), [9](#), [13](#), [17](#), [20](#), [22](#)

use\_onnx\_backend, [27](#)