

Package: ssdiff (via r-universe)

October 29, 2026

Title Supervised Semantic Differential

Version 0.1.0

Description Interpretable, embedding-based analysis of concept meaning in text. Builds per-document concept vectors from local contexts around seed words, learns a semantic gradient that predicts a numeric outcome or separates groups, and interprets it through the nearest neighbours of each pole, clusters of those neighbours, text snippets and per-document scores. The method is described in Plisiecki, Lenartowicz, Pokropek, Małyska and Flakus (2025) <[doi:10.31234/osf.io/gvrsb_v1](https://doi.org/10.31234/osf.io/gvrsb_v1)>. A port of the 'Python' package 'ssdiff'.

License GPL (>= 3)

URL <https://github.com/pawlenartowicz/ssdiff-r>

BugReports <https://github.com/pawlenartowicz/ssdiff-r/issues>

Depends R (>= 4.5.0)

Imports graphics, grDevices, jsonlite, nlme, stats, tools, utf8, utils

Suggests plskit (>= 0.7.1), fastglm (>= 0.6.2), testthat (>= 3.0.0)

Additional_repositories <https://pawlenartowicz.r-universe.dev>

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Repository <https://pawlenartowicz.r-universe.dev>

Date/Publication 2026-10-29 14:44:25 UTC

RemoteUrl <https://github.com/pawlenartowicz/ssdiff-r>

RemoteRef HEAD

RemoteSha a6ba4c2eb986feb73ec25794da70688b95ddbc9b

Contents

accessors	2
contrast	6
docs	7
evaluate_lexicon	8
fit_groups	9
fit_multipls	11
fit_pls	13
fit_regression	15
install_backends	18
lexicon_tokens	19
load_wordpack	20
outcomes	21
page_data	21
predict.ssd_result	23
print.ssd	24
print.ssd_embeddings	24
print.ssd_lexicon	25
print.ssd_snippets	25
print.wordpack	27
regression-generics	27
set_abtt	29
ssd	30
suggest_lexicon	31
sweep_table	33
with_lexicon	34
wordpack_cache	35
wordpack_lookup	35
wordpack_open	36
wordpack_read_rows	36
wordpack_selftest	37
wordpack_tokenize	37
Index	38

accessors	<i>Words, clusters and snippets of a result</i>
-----------	---

Description

Read a fitted gradient in the embeddings and in the texts. Each accessor computes its table when called; the tables are data frames with the columns of Python ssdiff’s views, in the same order.

Usage

```
words(result, side = "both", n = 20, unit = NULL)
```

```
clusters(  
  result,  
  side = "both",  
  topn = 100,  
  k = NULL,  
  k_min = 2,  
  k_max = 10,  
  min_cluster_size = 2,  
  random_state = 2137,  
  unit = NULL  
)
```

```
cluster_words(  
  result,  
  side = "both",  
  topn = 100,  
  k = NULL,  
  k_min = 2,  
  k_max = 10,  
  min_cluster_size = 2,  
  random_state = 2137,  
  unit = NULL  
)
```

```
snippets(  
  result,  
  side = "both",  
  top_per_side = 30,  
  min_cosine = NULL,  
  unit = NULL  
)
```

```
cluster_snippets(  
  result,  
  side = "both",  
  top_per_cluster = 100,  
  min_cosine = NULL,  
  topn = 100,  
  k = NULL,  
  k_min = 2,  
  k_max = 10,  
  min_cluster_size = 2,  
  random_state = 2137,  
  unit = NULL  
)
```

Arguments

result	A result of <code>fit_pls()</code> , <code>fit_multipls()</code> , <code>fit_regression()</code> , <code>fit_groups()</code> or <code>contrast()</code> .
side	"pos", "neg" or "both".
n	The number of words per side; NULL for all 100.
unit	The unit, see "Units".
topn	The number of nearest words per side that are clustered.
k	The number of clusters, or NULL to choose it by silhouette.
k_min, k_max	The range of k tried when k is NULL.
min_cluster_size	Clusters with fewer words are left out.
random_state	Seed of k-means, a whole number from 0 to $2^{53} - 1$.
top_per_side	The number of snippets per side; NULL for all.
min_cosine	The lowest cosine a snippet may have, or NULL.
top_per_cluster	The number of snippets per cluster.

Details

- `words()`: the nearest words of each pole, by cosine with the gradient: side, rank, word, cos_beta, contrast. Of the 100 nearest words per side ($n = \text{NULL}$), the first n .
- `clusters()`: k-means clusters of the topn nearest words of each pole, with the k in $k_{\text{min}}..k_{\text{max}}$ of the best silhouette unless k is given: cluster_id, side, size, coherence (the mean cosine of the words with their centroid), centroid_cos_beta, contrast, top_words (the five words nearest the centroid). The positive side's clusters come in descending, the negative side's in ascending order of centroid_cos_beta.
- `cluster_words()`: every word of those clusters: cluster_id, side, word, cos_centroid, cos_beta, contrast.
- `snippets()`: text windows scored by cosine with each pole. Each occurrence of a lexicon word gives one window, the SIF-weighted words within the text's window on each side; without a lexicon each sentence is one window. The top_per_side best per side, at least min_cosine.
- `cluster_snippets()`: windows scored against each cluster's centroid, the top_per_cluster best per cluster; cluster_id matches clusters() called with the same arguments. Without a lexicon each document is one window.

The snippet tables have snippet_id, side, doc_id, cosine, seed (the lexicon word), start_token_idx, end_token_idx, start_sent_idx, end_sent_idx, text_window, text_surface (the whole document), cluster_id and contrast. Printing clips the two text columns to 40 characters; the data frame keeps the full text.

Neighbour lists leave out the words the pack's neighbour filter matches, and search the 50,000 rows of the pack held in memory. `clusters()`, `cluster_words()` and `cluster_snippets()` with the same arguments give the same clusters: k-means draws from ssdiff's own generator, seeded by random_state.

A text that came in as token lists has no sentences, so its snippets raise. A `contrast()` has words and clusters only.

Ids count from 1: `doc_id` is the document's row in the text, and `cluster_id` the k-means cluster, from 1.

Value

A data frame.

Units

`unit` picks one unit of a result with several:

- `fit_groups()`: a pair, `c("g1", "g2")` in either order (the canonical names, or the group labels);
- `fit_multipls()`, and `fit_pls()` or `fit_regression()` with `ssd_gradients > 1`: a dimension, "dim-2" or 2;
- `fit_regression()` with several text terms: a term key such as "sexM:ssd(essay)", or its position in `names(res$units)`.

`unit = NULL` is the fit's own gradient for `fit_pls()` and `fit_regression()` with one text term, and the only pair of a two-group fit. Otherwise it stacks every unit: groups by contrast, dimensions with a `dim` column, text terms with a `term` column.

The embeddings come from the session (see `load_wordpack()`); after `readRDS()` in a new session the pack is loaded from the cache by its full name, or the accessor stops with a message naming it.

See Also

`docs()`, `doc_text()`, `outcomes()`

Examples

```
## Not run:
r <- fit_pls(s, "score", data = df)
words(r, n = 10)
clusters(r, side = "pos")
snippets(r, top_per_side = 5)
g <- fit_groups(s, "condition", data = df)
words(g, unit = c("g1", "g3"))

## End(Not run)
```

contrast

*The difference of two gradients***Description**

`contrast()` reads the direction in which two gradients differ: $\text{unit}(g1) - \text{unit}(g2)$, scaled to unit length, so that neither the outcomes' scales nor the backends enter the difference. The gradients can come from any fits on embeddings of the same dimension, for example `r1$gradient` and `r2$gradient`, or two units, `res$units[["sexM:ssd(essay)"]]``$gradient`.

Usage

```
contrast(g1, g2, emb, labels = c("g1", "g2"))
```

Arguments

<code>g1, g2</code>	Two numeric vectors of the embeddings' dimension.
<code>emb</code>	Embeddings from <code>load_wordpack()</code> , read for the words.
<code>labels</code>	The names of the two gradients.

Details

The result has words and clusters (`words()`, `clusters()`, `cluster_words()`) in `emb`, and no documents: `snippets()`, `cluster_snippets()`, `docs()` and `doc_text()` raise. There is no test; `cos_g1_g2` is the only statement about how the two gradients relate.

Value

An object of class `c("ssd_contrast", "ssd_result")`, a list with `beta` and `gradient` (both the unit difference), `stats` (a one-row data frame: `cos_g1_g2`, the cosine of the two gradients, and `diff_norm`, the length of $\text{unit}(g1) - \text{unit}(g2)$ before scaling), `labels`, and `texts` (one entry holding the pack of `emb`).

Examples

```
## Not run:
r1 <- fit_pls(s, "score", data = df)
r2 <- fit_pls(s, "age", data = df)
cc <- contrast(r1$gradient, r2$gradient, emb, labels = c("score", "age"))
words(cc)

## End(Not run)
```

docs

*Documents of a result***Description**

`docs()` gives one row per document a unit was fitted on. For `fit_pls()` and `fit_regression()`: `doc_id`, `y_true`, `y_hat`, `residual` and `alignment_score` (the cosine of the document's vector with the gradient). `y_hat` is the text's part of the prediction on the outcome's scale, or the model's fitted mean (on the response scale) for a model with several text terms, an interaction or a family other than the Gaussian; `residual` is `y_true` minus the model's fitted value, or for `fit_pls()` with a formula the PLS residual with the covariates taken out. For a pair of `fit_groups()`: `doc_id`, `group` (the label) and `alignment_score`, the pair's documents only. For a dimension (`fit_multipls()`, `ssd_gradients > 1`): `doc_id` and `alignment_score`.

Usage

```
docs(result, which = "all", n = 20, direction = "both", unit = NULL)
```

```
doc_text(result, doc_id, unit = NULL)
```

Arguments

<code>result</code>	A result of <code>fit_pls()</code> , <code>fit_multipls()</code> , <code>fit_regression()</code> , <code>fit_groups()</code> or <code>contrast()</code> .
<code>which</code>	"all", "pos", "neg" or "misdiagnosed".
<code>n</code>	The number of documents for which other than "all".
<code>direction</code>	For "misdiagnosed": "both", "over" or "under".
<code>unit</code>	The unit, see "Units".
<code>doc_id</code>	Document ids (rows of the text) among the unit's documents.

Details

`which = "pos"` and `"neg"` give the `n` documents with the highest and the lowest `y_hat` (the alignment score where there is no `y_hat`). `which = "misdiagnosed"` gives the `n` largest residuals: `direction = "both"` by absolute value, `"over"` the most negative (the model predicts too high), `"under"` the most positive.

`doc_text()` gives the text of documents of the unit: the raw string, or for a text from token lists the tokens joined by spaces.

`doc_id` is the document's row in the text, from 1.

Value

`docs()`: a data frame. `doc_text()`: a character vector.

See Also[words\(\)](#)**Examples**

```
## Not run:
r <- fit_pls(s, "score", data = df)
docs(r, "pos", n = 5)
docs(r, "misdiagnosed", direction = "over")
doc_text(r, 12)

## End(Not run)
```

evaluate_lexicon	<i>Evaluate a lexicon</i>
------------------	---------------------------

Description

Per-token statistics of a lexicon (as in [suggest_lexicon\(\)](#)) and its coverage of the documents.

Usage

```
evaluate_lexicon(
  s,
  y,
  lexicon,
  data = NULL,
  n_bins = 4,
  corr_cap = 0.3,
  var_type = "continuous"
)
```

Arguments

s	A text from ssd() .
y	The outcome, as in suggest_lexicon() ; NULL gives only the statistics that need no outcome, the others are NA.
lexicon	Character vector of lexicon words; a repeated word is evaluated once.
data	A data frame with one row per document, holding the column y names.
n_bins	Number of quantile bins of a continuous y.
corr_cap	Correlation at which the rank penalty is complete.
var_type	"continuous" or "categorical".

Value

An object of class `ssd_lexicon` with `stats`, `suggestions` (one row per word, sorted by rank, or by `freq` without `y`) and `summary`, a one-row data frame: `docs_any` (documents with any lexicon word), `cov_all` (their share), `q1` and `q4` (that share in the lowest and highest bin of `y`, or the lowest and highest group), `corr_any` (its correlation with `y`, or Cramér's V), `hits_mean`, `hits_median` (occurrences per document), `types_mean`, `types_median` (distinct words per document); for categorical `y` also `group_cov`, a list column holding the share per group as a named numeric vector.

Examples

```
## Not run:
ev <- evaluate_lexicon(s, "score", c("work", "job", "career"), data = df)
ev$summary

## End(Not run)
```

fit_groups

*Compare groups of documents***Description**

Tests whether groups of documents differ in meaning: each group's centroid is the unit-length mean of its document vectors, and two groups differ by $T = 1 - \cos$ of their centroids. One permutation test gives the omnibus p-value (for the mean T over all pairs) and a p-value for every pair, each $(\text{count}(\text{null} \geq \text{observed}) + 1) / (n_{\text{perm}} + 1)$. The pair p-values are corrected for multiple comparisons.

Usage

```
fit_groups(
  s,
  y,
  data = NULL,
  median_split = FALSE,
  n_perm = 5000,
  correction = "holm",
  random_state = 2137,
  verbose = FALSE
)
```

Arguments

<code>s</code>	A text from <code>ssd()</code> , with document vectors.
<code>y</code>	The group labels: a factor, character or numeric vector with one value per document, or the name of a column of data. With <code>median_split = TRUE</code> , a numeric outcome. Documents with NA or "" drop out.
<code>data</code>	A data frame with one row per document, or NULL.

median_split	TRUE splits a numeric y at its median.
n_perm	Number of permutations.
correction	Correction of the pair p-values: "holm", "bonferroni", "fdr_bh" (Benjamini-Hochberg) or "none", computed with <code>stats::p.adjust()</code> .
random_state	Seed of the permutations and of the median split's tie-break, a whole number from 0 to $2^{53} - 1$.
verbose	TRUE shows a progress bar and the omnibus p-value.

Details

Groups with fewer than 20 documents are dropped, each with a warning. The groups are named g1, g2, ... in sorted order of their labels (group_labels holds the labels), and every pair g1_g2, ... is a unit with its own gradient, the difference of the two groups' mean vectors.

With median_split = TRUE, y is numeric and is split at its median into "low" and "high". Documents at the median are shared out at random (with random_state) so that the two groups are as equal in size as possible. No group is dropped then.

The permutations come from ssdiff's own random generator, the one the Python package uses, so the same data and random_state give the same p-values in R and Python. R's `set.seed()` has no effect on them.

Value

An object of class `c("ssd_groups", "ssd_result")`, a list with:

G, n_kept the number of groups and of documents used;
group_labels a named character vector, g1, g2, ... to the labels;
groups the group (g1, ...) of each document used;
stats a one-row data frame: G, n_kept, n_perm, correction, random_state, pvalue (the omnibus p-value);
test a one-row data frame: name, pvalue, omnibus_T, omnibus_p, G, n_kept, n_perm, correction, random_state;
pairs a data frame with one row per pair: contrast, g1, g2, T, p_raw, p_corrected, cohens_d (on the documents' projections onto the pair's contrast), n_g1, n_g2, contrast_norm;
units one entry per pair, named by contrast, with g1, g2, beta (mean of g1 minus mean of g2), gradient (beta at unit length), beta_norm and alignment_scores (the cosine of each of the pair's documents with gradient);
texts the text the fit used, with its tokens, settings, pack, and the vectors (x) and rows (keep) of the documents used;
fit_args median_split, n_perm, correction, random_state.

Examples

```
## Not run:
s <- ssd(df$essay, lexicon = c("work", "job"))
g <- fit_groups(s, "condition", data = df)
```

```

g$pairs
h <- fit_groups(s, df$score, median_split = TRUE)

## End(Not run)

```

fit_multipls

*Fit PLS gradients for several outcomes***Description**

Fits PLS3 (PLS correlation): the singular value decomposition of the cross-covariance of the standardized document vectors and the standardized outcomes. Dimension i pairs a direction in embedding space ($U[, i]$) with outcome loadings ($V[, i]$). Each dimension is a unit, `units[["dim-1"]]`, ..., with its own gradient.

Usage

```

fit_multipls(
  s,
  y,
  data = NULL,
  ssd_gradients = NULL,
  rotate = NULL,
  outcome_groups = NULL,
  n_splits = 50,
  n_perm = 1000,
  random_state = 2137,
  verbose = FALSE
)

```

Arguments

<code>s</code>	A text from <code>ssd()</code> , with document vectors.
<code>y</code>	The outcomes, numeric: a matrix or data frame with one row per document and at least two columns, or column names of data. A row with a missing value in any outcome drops the document.
<code>data</code>	A data frame with one row per document, or NULL.
<code>ssd_gradients</code>	The number of dimensions, from 1 to $\min(q, n - 1, D)$ for q outcomes. NULL means q , or with <code>outcome_groups</code> the number of rotation target rows.
<code>rotate</code>	NULL, "words" or "outcomes".
<code>outcome_groups</code>	With <code>rotate = "outcomes"</code> only: a named list of character vectors of outcome names. Each group becomes one rotation target row, the mean of its outcomes' loadings; outcomes in no group keep their own row. <code>ssd_gradients</code> must not exceed the number of rows.
<code>n_splits</code>	Random splits of the test.

n_perm	Permutations of the test; unused when split_nb runs.
random_state	Seed of the test, a whole number from 0 to $2^{53} - 1$.
verbose	TRUE shows plskit's progress.

Details

Significance is plskit's PLS3 test of the first dimension, with test_method = "auto" (split_exact or split_nb, chosen from the vectors alone): one p-value for the fit, the same for every rotate. Dimensions have no p-value of their own.

Value

An object of class c("ssd_multipls", "ssd_result"), a list with:

stats a one-row data frame: n, q, ssd_gradients, rotate, cov_explained (the share of the squared cross-covariance the dimensions hold), pvalue;

pls_info the rotation's diagnostics: ssd_gradients, rotate, order, signs, kaiser_normalized, sweeps, V_converged, random_state;

test a one-row data frame: name (the method plskit ran), pvalue, statistic, n_splits, n_perm, random_state;

y, outcome_names, outcome_groups the outcomes of the rows used, their names, and the groups;

U, V, singular_values, U_rot, V_rot, R, x_scores, y_scores the decomposition, its rotation, and the scores of the standardized vectors and outcomes on the rotated dimensions;

units one per dimension, keyed "dim-1", ..., each with beta (U_rot[, i] in the scale of the document vectors), gradient and stats (cov_share, score_corr);

texts the text the fit used, with its tokens, settings, pack, and the vectors (x) and rows (keep) of the documents used;

fit_args the fit's arguments.

Rotation

One orthogonal R is applied to both sides ($U_{rot} = U \%*\% R$, $V_{rot} = V \%*\% R$):

- NULL: the dimensions of the decomposition as they are.
- "words": varimax towards the projection of the 50,000 most frequent words of the pack onto U.
- "outcomes": varimax towards the outcome loadings V, or towards one row per group of outcome_groups. With ssd_gradients equal to the number of outcomes and no groups, $R = t(V)$: each dimension loads 1 on one outcome.

The rotated modes order the dimensions by their squared cross-covariance. Every mode signs each dimension so that its largest outcome loading is positive.

Examples

```
## Not run:
s <- ssd(df$essay, lexicon = c("work", "job"))
r <- fit_multipls(s, c("valence", "arousal"), data = df)
r <- fit_multipls(s, c("v1", "v2", "a1"), data = df, rotate = "outcomes",
  outcome_groups = list(valence = c("v1", "v2")))
r$units[["dim-1"]]$stats

## End(Not run)
```

fit_pls

Fit a PLS gradient

Description

Fits a partial least squares regression (PLS1) of a numeric outcome on the text's document vectors, both standardized, and turns its coefficients into a semantic gradient (beta): the direction in embedding space along which the outcome grows. beta is oriented so that a higher alignment means a higher outcome.

Usage

```
fit_pls(
  s,
  y,
  data = NULL,
  k = "auto",
  k_max = 5,
  ssd_gradients = 1,
  rotate = "words",
  n_splits = 50,
  random_state = 2137,
  test = TRUE,
  verbose = FALSE
)
```

Arguments

s	A text from <code>ssd()</code> , with document vectors.
y	The numeric outcome: values, one per document, a column name of data, or a formula whose left side names the outcome column and whose right side holds fixed covariates. Documents with a missing outcome drop out.
data	A data frame with one row per document, or NULL.
k	The number of PLS components, or "auto".
k_max	The largest k that k = "auto" tries, capped at n - 1 and the vector dimension.
ssd_gradients	The number of gradients, from 1 to min(n - 1, D).

rotate	"words" or NULL: the rotation of the gradients when <code>ssd_gradients > 1</code> .
n_splits	Random splits of the split-half tests.
random_state	Seed of <code>pls</code> ’s random splits, a whole number from 0 to $2^{53} - 1$.
test	FALSE skips the confirmatory test; pvalue is then NA.
verbose	TRUE shows the steps.

Details

With `k = "auto"`, `pls`’s `pls1_find_k_optimal()` (selector "r2_se", five-fold cross-validation) picks the number of components, and the p-value is `pls`’s confirmatory test at `k = 1`, independent of that choice. With a whole number `k`, the model has `k` components and the test runs at `k`. The test is `pls`’s `test_method = "auto"` (Lenartowicz and Plisiecki, 2026), which runs `split_exact` or `split_nb`, chosen from the vectors alone; `test$name` says which ran.

Value

An object of class `c("ssd_pls", "ssd_result")`, a list with:

`stats` a one-row data frame: backend ("PLS"), `r2`, `pvalue`, `n_raw`, `n_kept`, `n_dropped`, `y_mean`, `y_std`, `beta_norm`, `delta`, `iqr_effect`, `y_corr_pred`;

`fit_info` a one-row data frame of the fit’s settings, the number of components (`n_components`) and the `k` of the test (`p_at_k`) included;

`test` a one-row data frame: `name` (the method `pls` ran), `pvalue`, `split_r2`, `n_splits`, `random_state`;

`beta`, `gradient` the gradient, on the standardized-outcome scale, and its unit vector;

`y`, `alignment_scores`, `y_hat`, `residuals` the outcome of the rows used, their cosine with the gradient, the predicted outcome and the residual (with the covariates taken out, given a formula);

`outcome_name` the outcome’s name;

`component_scores`, `component_weights` the PLS scores `T` and weights `W`;

`find_k` `pls`’s `k`-selection result for `k = "auto"`, else NULL;

`pls_info`, `units` with `ssd_gradients > 1`: the rotation’s diagnostics (`ssd_gradients`, `rotate`, `order`, `signs`, `kaiser_normalized`, `sweeps`, `V_converged`, `random_state`), and one unit per gradient, keyed "dim-1", ..., each with `beta`, `gradient` and `stats` (`cov_share`, `score_corr`); else NULL;

`texts` the text the fit used, with its tokens, settings, `pack`, and the vectors (`x`) and rows (`keep`) of the documents used;

`fit_args` the formula (as a string) and the other arguments.

Covariates

With a formula such as `score ~ age + gender`, the outcome and every column of the document vectors are first regressed on the formula’s fixed effects by least squares, and PLS runs on the residuals. `r2` is then the residualized outcome’s, and the residuals the fit reports have the covariates taken out. The covariates’ own coefficients are not reported: use `fit_regression()` for them, and for random effects, which `fit_pls()` does not take.

Several gradients

With `ssd_gradients = N > 1`, the result also holds N gradients, `units[["dim-1"]] ... units[["dim-N"]]`, from the weight vectors of an N -component PLS fit on the same data. With `rotate = "words"` they are varimax-rotated towards the projection of the 50,000 most frequent words of the pack, ordered by the correlation of their scores with the outcome, and signed so that it is positive. `rotate = NULL` keeps them in PLS order. The standard fit does not change.

Examples

```
## Not run:
s <- ssd(df$essay, lexicon = c("work", "job"))
r <- fit_pls(s, "score", data = df)
r <- fit_pls(s, score ~ age + gender, data = df, k = 2)
r <- fit_pls(s, df$score, ssd_gradients = 3)
r$units[["dim-1"]]$stats

## End(Not run)
```

fit_regression

Regress an outcome on texts, with covariates

Description

Fits a regression of the outcome on the first K principal components of the text's document vectors, together with the covariates of a formula, and turns the components' coefficients back into a semantic gradient (beta): the direction in embedding space along which the outcome changes.

Usage

```
fit_regression(
  s,
  y,
  data = NULL,
  family = gaussian(),
  fixed_k = NULL,
  k_min = 2,
  k_max = 120,
  k_step = 2,
  ssd_gradients = 1,
  verbose = FALSE,
  sweep_args = NULL
)
```

Arguments

<code>s</code>	A text from <code>ssd()</code> ; a named list of texts; or the formula, when every text comes from a column of data (then leave out <code>y</code>).
----------------	---

<code>y</code>	The outcome: numeric values, one per document, a column name of data, or a formula whose left side names the outcome column, such as <code>score ~ age + sex * ssd() + (1 school)</code> . Documents with a missing outcome drop out.
<code>data</code>	A data frame with one row per document, or NULL.
<code>family</code>	A family as <code>stats::glm()</code> takes it: a family object (<code>binomial(link = "probit")</code>), a family function or its name.
<code>fixed_k</code>	A fixed number of components, or NULL for the sweep.
<code>k_min, k_max, k_step</code>	The K values the sweep tries.
<code>ssd_gradients</code>	The number of gradients, from 1 to K. With $N > 1$ (one text, one plain <code>ssd()</code> term), the K components are split into N consecutive blocks; each block's part of beta is one gradient, and the N gradients are varimax-rotated towards the pack's most frequent words as in <code>fit_pls()</code> . They are the units "dim-1", ..., with beta, gradient and stats (<code>cov_share</code> , <code>score_corr</code>), and <code>pls_info</code> holds the rotation's diagnostics. The standard fit does not change. Needs <code>plsfit</code> .
<code>verbose</code>	TRUE shows the sweep's progress.
<code>sweep_args</code>	A named list of sweep settings: <code>cluster_topn</code> (100), <code>cluster_k_min</code> (2), <code>cluster_k_max</code> (5), <code>weight_by_size</code> (TRUE), <code>auck_radius</code> (3).

Value

An object of class `c("ssd_pca_ols", "ssd_result")`, a list with:

`stats` a one-row data frame: backend ("PCA+OLS" for a Gaussian identity model without random effects, else "PCA+GLMM"), `r2`, `r2_adj` (the whole model's; NA for GLMs and mixed models), `pvalue` (the test of all text terms), `n_raw`, `n_kept`, `n_dropped`, `y_mean`, `y_std`, `beta_norm`, `delta`, `iqr_effect`, `y_corr_pred`;

`fit_info` a one-row data frame of the fit's settings, K (`n_components`) included;

`test` one row per test: label, name, pvalue, statistic, df, df_resid, `r2_delta`;

`beta`, `gradient` the gradient (on the standardized-outcome scale for a Gaussian identity model, else on the link scale) and its unit vector; NULL with several units;

`y`, `alignment_scores`, `y_hat` the outcome of the rows used, their cosine with the gradient, and the predicted outcome;

`outcome_name`, `formula`, `pca_weights` the outcome's name, the formula as written (NULL without one), and the component weights;

`model` the `lm`, `glm` or `fastglm` fit;

`covariates` the covariates' estimates with the text held fixed: term, estimate, se, statistic, `p_value`, `ci_low`, `ci_high`;

`terms` a Wald test per unit: term, name, statistic, df, df_resid, pvalue;

`model_test` fvalue, f_pvalue (OLS) or llr, llr_pvalue: the whole model against outcome ~ 1;

`units` with several text terms, one entry per term, keyed as "sexM:ssd(essay)", with its own `beta`, `gradient`, `pca_weights`, `alignment_scores`, stats, `fit_info`, text (its position in texts) and test (the label of the test row of its term); with `ssd_gradients > 1`, one entry per gradient, keyed "dim-1", ..., with beta, gradient and stats (`cov_share`, `score_corr`); else NULL;

pls_info with `ssd_gradients > 1`, the rotation's diagnostics (`ssd_gradients`, `rotate`, `order`, `signs`, `kaiser_normalized`, `sweeps`, `V_converged`, `random_state`); else `NULL`;

texts one entry per text: its tokens, settings, the vectors and rows used, `k`, `pca_components` and `sweep`;

obs_names the names of the rows used: the row names of data, or the row numbers without data;

fit_args the formula (as a string) and the other arguments.

The model generics `coef()`, `vcov()`, `confint()`, `nobs()`, `logLik()`, `residuals()`, `fitted()`, `df.residual()` and `ranef()` read the model, without the text's coefficients.

The `ssd()` term

The text is a term of the formula, written `ssd()`: `score ~ age + ssd()`. A formula without an `ssd()` term gets `+ ssd()` appended, and without a formula the model is the intercept plus the text. The term can interact with covariates: in `score ~ age + sex * ssd()` (treatment coding), `ssd()` is the gradient at the reference level of `sex`, and `sexM:ssd()` the difference between level `M` and the reference, so the gradient at `M` is the sum of the two. `score ~ sex:ssd()` has one gradient per level, and in `score ~ age * ssd()` `ssd()` is the gradient at `age = 0`. Each term is then a unit of the result (`res$units`, keyed "`sexM:ssd()`", ...). A formula with `ssd(...)` is used as written, so `~ sex:ssd()` has no main text term. `ssd()` cannot appear inside a random-effect term.

The term is never evaluated: `fit_regression()` reads it from the formula and replaces it with `(.ssd_pca_1 + ... + .ssd_pca_K)` before a model function sees the formula. Names starting with `.ssd_` are reserved.

Several texts

`s` can be a named list of texts, each named in the formula as `ssd(name)`: `fit_regression(list(essay = s1, diary = s2), score ~ age + ssd(essay) + sex * ssd(diary), data = df)`. A name that is not an element of `s` is built from the column of that name in data as `ssd(data[[name]])`, with every default and one `load_wordpack()` for all such texts. When every text comes from a column, the formula can come first: `fit_regression(score ~ ssd(essay), data = df)`. A row stays when it has a valid outcome and a vector in every text. Each text has its own standardization, PCA and `K` (`fixed_k` applies to all).

Choosing `K`

Unless `fixed_k` is given, `K` is chosen once per text by a sweep over `seq(k_min, k_max, k_step)`: OLS of the standardized outcome on the standardized components, whatever the family, scoring each `K` by the interpretability of the clusters of neighbours of both poles and by the stability of the gradient, and picking the best joint score. The per-`K` table is `res$texts[[i]]$sweep`.

Two model paths

A model without random effects, Gaussian with the identity link, is fitted with `stats::lm()`; binomial and Poisson models without random effects with `stats::glm()`. A formula with random effects (`(1 | g)`) or another family or link is fitted with the `fastglmm` package, which must be installed. Both paths give the numbers of Python's `ssdiff`.

fastglmm takes column names of letters, digits and underscores only, starting with a letter: a formula column such as `age.years` raises on that path (rename it), and data columns starting with `ssid__` are reserved. There the text's columns are passed as `ssid__pca_1`, ..., so the coefficient names of `res$model` use `ssid__` where everything else in the result shows `.ssid_`.

Tests

`res$test` holds a refit test of all text terms (an F test for OLS, else a likelihood-ratio test; on fastglmm as fastglmm reports it), with several texts one refit test per text, and with a text that has several terms a Wald test per term label (`sex:ssd()` tests whether the gradient differs by sex). `res$terms` has a Wald test per unit, and `res$model_test` the whole model against outcome `~ 1`.

Examples

```
## Not run:
s <- ssid(df$essay, lexicon = c("work", "job"))
r <- fit_regression(s, "score", data = df)
r <- fit_regression(s, score ~ age + sex * ssid(), data = df)
r$test
coef(r)
r <- fit_regression(score ~ ssid(essay) + ssid(diary), data = df)
if (requireNamespace("fastglmm", quietly = TRUE)) {
  r <- fit_regression(s, score ~ age + (1 | school), data = df)
}

## End(Not run)
```

install_backends

Install the plsKit and fastglmm backends

Description

Installs the packages that the PLS fits and the mixed-model path of `fit_regression()` need. Both are on r-universe, not CRAN. Run it once.

Usage

```
install_backends(pkgs = c("plsKit", "fastglmm"), ...)
```

Arguments

<code>pkgs</code>	Packages to install.
<code>...</code>	Passed to <code>utils::install.packages()</code> . When it holds lib, the load check on Linux looks there.

Details

On Windows and macOS the r-universe repository serves binaries. On Linux the prebuilt binary for Ubuntu `resolute` is tried first; a package whose binary installs but does not load (for example on a system with `glibc` older than 2.39) is then built from source, which needs Rust (<https://rustup.rs>).

Value

NULL, invisibly, on Linux; elsewhere the value of `utils::install.packages()`.

Examples

```
## Not run:  
install_backends()  
  
## End(Not run)
```

lexicon_tokens	<i>Tokens of a lexicon suggestion</i>
----------------	---------------------------------------

Description

Tokens of a lexicon suggestion

Usage

```
lexicon_tokens(lex)
```

Arguments

`lex` A result of `suggest_lexicon()` or `evaluate_lexicon()`.

Value

The token column of `lex$suggestions`, ready to pass to `ssd()` or `with_lexicon()`.

load_wordpack	<i>Load a wordpack</i>
---------------	------------------------

Description

Loads word embeddings from a .wordpack file, by tag, by file or by the pack's full name. x is read in this order:

Usage

```
load_wordpack(x = "default")
```

Arguments

x A tag, a path to a .wordpack file, or a pack's full name.

Details

1. A tag (a string with no dot): "default", "en", "en-dolma" or "en-wiki". The tag's message is shown and its pack is loaded.
2. A path where a file exists. It must end in .wordpack.
3. A full name such as "en-glove2024-dolma-1": the pack is loaded from the cache ([wordpack_cache\(\)](#)) and downloaded there first if it is missing. A cached pack loads with no network request.

The pack must pass its self-test. It loads at ABTT 1 with unit rows. The $\min(n, 50000)$ most frequent rows are kept in memory (about 120 MB for a 300-dimensional pack); other rows are read from the file when a text needs them, so the file must stay in place. Each loaded pack stays in a session registry, from which saved results find it again, so its memory is freed only when the R session ends, not by `rm()`.

Value

An object of class `ssd_embeddings`, a list with pack (the open [wordpack](#)), name, version, language, abtt, vectors (the in-memory rows, unit length), stopwords and `neighbor_filter` (a regular expression for words hidden from neighbour lists).

See Also

[set_abtt\(\)](#)

Examples

```
## Not run:
emb <- load_wordpack()           # the default English pack
emb <- load_wordpack("en-wiki")
emb <- load_wordpack("path/to/pack.wordpack")

## End(Not run)
```

outcomes

Outcome loadings of a multi-outcome fit

Description

The loadings of the outcomes on each dimension of `fit_multipls()` (`V_rot`): outcome, group (the outcome's group of `outcome_groups`, or NA) and loading. With `outcome_groups`, one row per group follows, with the plain mean of its outcomes' loadings (outcomes scored in opposite directions cancel).

Usage

```
outcomes(result, unit = NULL)
```

Arguments

<code>result</code>	A result of <code>fit_multipls()</code> .
<code>unit</code>	A dimension, "dim-2" or 2; NULL stacks every dimension with a <code>dim</code> column.

Value

A data frame.

Examples

```
## Not run:
r <- fit_multipls(s, c("valence", "arousal"), data = df)
outcomes(r, "dim-1")

## End(Not run)
```

page_data

The results page

Description

`save_page()` writes a result as one self-contained HTML file: the results page of Python `ssdiff`, which opens from disk in a browser, with no network. It shows the words, clusters, snippets and documents of every unit of the result, its tests and its settings. `page_data()` gives the data the page reads, as a nested list.

Usage

```

page_data(
  result,
  title = NULL,
  dataset = NULL,
  snippets_per_side = 200,
  snippets_per_cluster = 50,
  doc_texts = 50
)

save_page(
  result,
  path = "ssd_results.html",
  title = NULL,
  dataset = NULL,
  snippets_per_side = 200,
  snippets_per_cluster = 50,
  doc_texts = 50
)

```

Arguments

result	A result of <code>fit_pls()</code> , <code>fit_multipls()</code> , <code>fit_regression()</code> , <code>fit_groups()</code> or <code>contrast()</code> .
title	The page's title; NULL for "SSD results".
dataset	A named list or named character vector describing the data, shown on the page; NULL for none.
snippets_per_side	The number of snippets per side of each unit.
snippets_per_cluster	The number of snippets per cluster.
doc_texts	The number of documents at each end of each ranking whose text goes into the page; Inf for every document, 0 for none.
path	The file to write.

Details

Every table is computed when the page is written, with the accessors' default cluster arguments (`clusters()`). A text that came in as token lists has no snippet tables, and its documents' text is the tokens joined by spaces.

Of the documents, the page keeps the text of the `doc_texts` highest and lowest by alignment score, and for `fit_pls()` and `fit_regression()` also the `doc_texts` most over- and under-predicted; the other documents are listed without text.

In `page_data()`, a JSON object is a named list and a JSON array an unnamed list or an `I()` vector. Every table is `list(columns = I(names(df)), rows = df)`, with `df` a data frame. Ids are as in the page: `cluster_id` and a unit's text count from 0, as the page counts them; `doc_id` is the document's row in the text, from 1.

Value

page_data(): a named list. save_page(): path, invisibly.

Examples

```
## Not run:
r <- fit_regression(s, "score", data = df)
save_page(r, "results.html", title = "Essays", dataset = c(File = "essays.csv"))
str(page_data(r), max.level = 1)

## End(Not run)
```

predict.ssd_result	<i>Predict the outcome of new texts</i>
--------------------	---

Description

Builds the new documents' vectors as the fit built its own: the same pack, lexicon, window, sif_a and mode, with SIF weights from the training documents' word counts. The prediction is the text's part on the outcome's scale, $y_{\text{mean}} + y_{\text{std}} * (x - x_{\text{mean}}) \% \% \text{beta}$, so on the training texts it equals `res$y_hat`.

Usage

```
## S3 method for class 'ssd_pls'
predict(object, newdata, ...)

## S3 method for class 'ssd_pca_ols'
predict(object, newdata, ...)
```

Arguments

object	A result of <code>fit_pls()</code> or <code>fit_regression()</code> .
newdata	Texts in the form of the fit's text: a character vector (tokenized with the result's pack) for a fit on strings, a list of token lists for a fit on token lists.
...	Unused.

Details

Only fits without a formula, and for `fit_regression()` only the Gaussian family with the identity link: another model's prediction needs the new documents' covariates and random effects, or the model's intercept and inverse link.

Value

A numeric vector with one prediction per text; NA for a text that gets no vector (no word in the pack, or no lexicon word).

Examples

```
## Not run:
r <- fit_pls(s, "score", data = df)
predict(r, c("A new essay about work.", "Another one."))

## End(Not run)
```

print.ssd	<i>Print a text</i>
-----------	---------------------

Description

Shows the number of documents and of documents with a vector, the vector dimension, the lexicon size, the tokenizer and the fit functions.

Usage

```
## S3 method for class 'ssd'
print(x, ...)
```

Arguments

x	A text from ssd() .
...	Unused.

Value

x, invisibly.

print.ssd_embeddings	<i>Print loaded embeddings</i>
----------------------	--------------------------------

Description

Shows the pack's full name, its number of words and dimensions, the ABTT and the file path.

Usage

```
## S3 method for class 'ssd_embeddings'
print(x, ...)
```

Arguments

x	Embeddings from load_wordpack() .
...	Unused.

Value

x, invisibly.

print.ssd_lexicon	<i>Print a lexicon suggestion</i>
-------------------	-----------------------------------

Description

Shows the stats, the first 20 suggestions and, from `evaluate_lexicon()`, the coverage summary.

Usage

```
## S3 method for class 'ssd_lexicon'
print(x, ...)
```

Arguments

x	A result of <code>suggest_lexicon()</code> or <code>evaluate_lexicon()</code> .
...	Unused.

Value

x, invisibly.

print.ssd_snippets	<i>Print and summarise a result</i>
--------------------	-------------------------------------

Description

`print()` shows a result's headline (r^2 , p-value, n, backend; for groups the omnibus test) and lists the accessors and fields to read next. `summary()` prints a longer account and returns its tables invisibly:

Usage

```
## S3 method for class 'ssd_snippets'
print(x, ...)

## S3 method for class 'ssd_pls'
print(x, ...)

## S3 method for class 'ssd_pca_ols'
print(x, ...)

## S3 method for class 'ssd_groups'
print(x, ...)
```

```
## S3 method for class 'ssd_multipls'
print(x, ...)

## S3 method for class 'ssd_contrast'
print(x, ...)

## S3 method for class 'ssd_pls'
summary(object, ...)

## S3 method for class 'ssd_groups'
summary(object, ...)

## S3 method for class 'ssd_multipls'
summary(object, ...)

## S3 method for class 'ssd_contrast'
summary(object, ...)

## S3 method for class 'ssd_pca_ols'
summary(object, ...)
```

Arguments

x, object	A result, or for print() a snippet table.
...	Unused.

Details

- `fit_pls()`: the number of components, the split-half test and the R^2 (the partial R^2 with a formula); returns stats and test.
- `fit_regression()`: the formula, the covariates' coefficients (estimate, standard error, t or z, p), one Wald test per text term, with several text terms the joint tests (all text terms; each text) with the R^2 they add (OLS only), the fit statistics, and the test of the whole model against the intercept-only model: the F test for OLS, as `lm()` prints it, else the likelihood-ratio test. It returns a list with coefficients (the covariates' matrix, as `coef()` and `vcov()` give them), terms and model_test. `summary(res$model)` gives the model package's own summary.
- `fit_groups()`: the omnibus test and the pairs; returns test and pairs.
- `fit_multipls()`: the stats and each dimension's cov_share and score_corr; returns stats and dims.
- `contrast()`: the two labels, cos_g1_g2 and diff_norm; returns stats.

The tables of `snippets()` and `cluster_snippets()` print with `text_window` and `text_surface` clipped to 40 characters; the data frames keep the full text.

Value

`print()`: x, invisibly. `summary()`: a list, invisibly.

Examples

```
## Not run:
r <- fit_regression(s, score ~ age + ssd(), data = df)
r
summary(r)

## End(Not run)
```

print.wordpack	<i>Print a wordpack</i>
----------------	-------------------------

Description

One line: name, version, number of keys, dimensions and language.

Usage

```
## S3 method for class 'wordpack'
print(x, ...)
```

Arguments

x	A wordpack from wordpack_open() .
...	Unused.

Value

x, invisibly.

regression-generics	<i>Model generics of a regression result</i>
---------------------	--

Description

A [fit_regression\(\)](#) result reads like the model it holds (res\$model). A text term has K component coefficients and no single estimate, so these hold the covariates only: the model's coefficients without the .ssd_* ones. The text terms are tested in res\$terms and res\$test, and the whole model in res\$model_test.

Usage

```
## S3 method for class 'ssd_pca_ols'
coef(object, ...)

## S3 method for class 'ssd_pca_ols'
vcov(object, ...)

## S3 method for class 'ssd_pca_ols'
confint(object, parm, level = 0.95, ...)

## S3 method for class 'ssd_pca_ols'
nobs(object, ...)

## S3 method for class 'ssd_pca_ols'
df.residual(object, ...)

## S3 method for class 'ssd_pca_ols'
logLik(object, ...)

## S3 method for class 'ssd_pca_ols'
residuals(object, ...)

## S3 method for class 'ssd_pca_ols'
fitted(object, ...)

## S3 method for class 'ssd_pca_ols'
ranef(object, ...)
```

Arguments

object	A result of fit_regression() .
...	Unused.
parm	Covariates (names or positions among them); all by default.
level	The confidence level.

Details

`confint()` gives Wald intervals on every model path (t on the residual df for `lm`, normal for `glm`, as `fastglmm` reports them), the intervals of `res$covariates`. `df.residual()` is `nobs()` minus the fixed coefficients that are not aliased, for every model (for a mixed model this is not a Satterthwaite df). `logLik()` carries df and nobs, so [AIC\(\)](#) and [BIC\(\)](#) work. `residuals()` (response residuals) and `fitted()` (fitted means on the response scale) are named by the row names of data of the rows used, or by the row numbers without data. `ranef()` gives one data frame per grouping factor, one row per level and one column per random-effect term; an empty list without random effects.

Value

`coef()`: named estimates. `vcov()`: their covariance matrix. `confint()`: a matrix with the lower

and upper limits. `nobs()`, `df.residual()`: numbers. `logLik()`: a logLik object. `residuals()`, `fitted()`: named vectors. `ranef()`: a named list of data frames.

Examples

```
## Not run:
r <- fit_regression(s, score ~ age + sex * ssd(), data = df)
coef(r)
confint(r)
AIC(r)

## End(Not run)
```

set_abtt

Set the ABTT of loaded embeddings

Description

ABTT ("all-but-the-top") removes the pack's mean and its top `m` principal components from every row, then scales each row to unit length again. A pack stores its rows before ABTT, so any `m` from 0 to 10 can be set, up or down; the rows are read again from the file.

Usage

```
set_abtt(emb, m)
```

Arguments

<code>emb</code>	Embeddings from <code>load_wordpack()</code> .
<code>m</code>	Number of components to remove, a whole number from 0 to 10.

Value

New embeddings at ABTT `m`; `emb` is unchanged.

Examples

```
## Not run:
emb <- load_wordpack()
emb3 <- set_abtt(emb, 3)

## End(Not run)
```

ssd

*Build a text for SSD analysis***Description**

ssd() tokenizes a text once and builds its document vectors. The result knows nothing about an outcome: every fit ([fit_pls\(\)](#), [fit_multipls\(\)](#), [fit_regression\(\)](#), [fit_groups\(\)](#)) takes it first and the outcome second.

Usage

```
ssd(
  texts,
  embeddings = NULL,
  lexicon = NULL,
  window = 3,
  sif_a = 0.001,
  use_full_doc = NULL
)
```

Arguments

texts	A character vector (or a factor), one text per document (NA is an empty document), tokenized with the pack's tokenizer and stopwords. Or a list of character vectors, one token list per document, used as given.
embeddings	Embeddings from load_wordpack() . NULL loads load_wordpack() , the "default" tag.
lexicon	Character vector of lexicon words, or NULL.
window	Context window: the number of words on each side of a lexicon word, a whole number of at least 1.
sif_a	SIF smoothing parameter, greater than 0. NULL weighs every word 1 (a plain mean).
use_full_doc	TRUE builds each document's vector from all its words, FALSE from the windows around the lexicon's words. NULL means TRUE without a lexicon and FALSE with one.

Details

A document's vector is built from its words' embeddings, each weighted by SIF, $sif_a / (sif_a + count / total)$, with word counts over all documents of the text. With a lexicon (`use_full_doc = FALSE`), each occurrence of a lexicon word gives the weighted mean of the words within window on each side, the occurrence itself left out, and the document's vector is the mean of those. With `use_full_doc = TRUE` it is the weighted mean of all the document's words. A document with no vector is not kept; the others are scaled to unit length.

Without a lexicon and with `use_full_doc = FALSE`, the object holds tokens only. [suggest_lexicon\(\)](#) and [evaluate_lexicon\(\)](#) work on it, which makes it the cheap way to tokenize before choosing a lexicon; [with_lexicon\(\)](#) then adds the vectors. The fits need vectors and raise on it.

Value

An object of class `ssd`, a list with `docs` (the tokens of each document), `pre_docs` (sentence structure for snippets, `NULL` for token lists), `tokenizer`, `x` (one unit row per document with a vector, `NULL` for tokens only), `keep` (whether each document has a vector), `n_docs`, `n_kept`, `lexicon`, `window`, `sif_a`, `use_full_doc`, and internally the SIF counts and a reference to the pack (`pack`).

Two uses of the name `ssd`

`ssd()` is also the text term in a `fit_regression()` formula, as in `score ~ age + sex * ssd()` or `score ~ ssd(essay)`. There it is never called: `fit_regression()` reads the term from the formula and replaces it with the text's principal components before any model function sees the formula. Passing such a formula to `lm()` or `glm()` directly calls this constructor, which fails on its missing texts.

See Also

`with_lexicon()`, `suggest_lexicon()`

Examples

```
## Not run:
emb <- load_wordpack()
s <- ssd(df$essay, embeddings = emb)                # full documents
s <- ssd(df$essay, embeddings = emb, lexicon = c("work", "job"))
s0 <- ssd(df$essay, embeddings = emb, use_full_doc = FALSE) # tokens only
s1 <- with_lexicon(s0, c("work", "job"))

## End(Not run)
```

suggest_lexicon	<i>Suggest lexicon words</i>
-----------------	------------------------------

Description

Ranks the tokens of a text as candidate lexicon words, or lists the tokens close to seed words. Only the tokens are needed, so `s` may be a tokens-only text (`ssd(texts, use_full_doc = FALSE)`).

Usage

```
suggest_lexicon(
  s,
  y = NULL,
  data = NULL,
  seeds = NULL,
  top_k = 30,
  min_docs = 5,
  n_bins = 4,
  corr_cap = 0.3,
```

```
    var_type = "continuous"
  )
```

Arguments

<code>s</code>	A text from <code>ssd()</code> .
<code>y</code>	The outcome: one value per document (numbers for continuous, labels for categorical), or a column name of data. Rows where it is missing are dropped. NULL is allowed with seeds: the outcome columns are then NA.
<code>data</code>	A data frame with one row per document, holding the column <code>y</code> names.
<code>seeds</code>	Character vector of seed words, each a key of the embeddings. A seed need not occur in the text.
<code>top_k</code>	The number of tokens returned; with seeds, the number of neighbours listed per seed.
<code>min_docs</code>	Minimum number of documents a token must be in.
<code>n_bins</code>	Number of quantile bins of a continuous <code>y</code> .
<code>corr_cap</code>	Correlation at which the rank penalty is complete.
<code>var_type</code>	"continuous" or "categorical".

Details

Without seeds, every token in at least `min_docs` documents is ranked by balanced coverage (the mean coverage over quantile bins of `y`, or over groups for categorical `y`) times $1 - \min(1, |\text{corr}| / \text{corr_cap})$, where `corr` is the token's correlation with `y` (Cramér's *V* for categorical `y`). Tokens that cover the documents evenly and do not track the outcome rank highest. `pvalue` is the point-biserial test (continuous) or the chi-square test (categorical, with Yates' correction for 2 groups).

With seeds, each seed lists itself, its `top_k` closest candidate tokens by cosine in the embeddings `s` was built on, and its forms with an English ending (`-s`, `-es`, `-d`, `-ed`, `-ing`, and `-ies`, `-ied` after `y`). A token appears once, under its closest seed, and rows are sorted by cosine. A lexicon matches exact tokens, so a plural counts only when it is listed.

Value

An object of class `ssd_lexicon`, a list with stats (one-row data frame: `var_type`, `n_docs`, `n_tokens`) and suggestions (data frame: `token`, `freq`, `cov_all`, `cov_bal`, `corr`, `pvalue`, `direction`, `rank`; with seeds also `seed` first and `cos` after `token`).

See Also

`evaluate_lexicon()`, `lexicon_tokens()`

Examples

```
## Not run:
s <- ssd(df$essay, use_full_doc = FALSE)
lex <- suggest_lexicon(s, "score", data = df)
lex <- suggest_lexicon(s, seeds = c("work", "job"))
```

```
s2 <- with_lexicon(s, lexicon_tokens(lex))

## End(Not run)
```

sweep_table	<i>The K sweep of a regression</i>
-------------	------------------------------------

Description

sweep_table() gives the per-K table of [fit_regression\(\)](#)'s sweep for one text: k, var_explained, the interpretability columns, the gradient's change from the previous K (beta_delta_1_minus_cos), their z-scores and window means, and joint_score, which picked K. It has no rows when K came from fixed_k.

Usage

```
sweep_table(result, text = NULL)

plot_sweep(result, text = NULL, path = NULL, dpi = 300)
```

Arguments

result	A result of fit_regression() .
text	A text's name or position; NULL when the fit has one text.
path	A file path for a PNG, or NULL.
dpi	The resolution of the PNG.

Details

plot_sweep() draws the chart of the sweep, in base graphics: the detrended interpretability (z) against K on the left axis, the change of the gradient (1 - cosine, a running median over 7 values of K) on the right axis, and the chosen K as a vertical line. With path it writes a PNG file at dpi, 6.4 by 4.8 inches; otherwise it draws on the current device.

Value

sweep_table(): a data frame. plot_sweep(): NULL, invisibly.

Examples

```
## Not run:
r <- fit_regression(s, "score", data = df)
sweep_table(r)
plot_sweep(r)
plot_sweep(r, path = "sweep.png")

## End(Not run)
```

with_lexicon	<i>Give a text another lexicon</i>
--------------	------------------------------------

Description

A new `ssd()` object on the same tokens, with new document vectors. The defaults are `ssd()`'s, not `s`'s settings. `ssd(texts, emb, lexicon)` equals `with_lexicon(ssd(texts, emb), lexicon)`, so a text can be tokenized once and fitted with several lexicons.

Usage

```
with_lexicon(s, lexicon, window = 3, sif_a = 0.001, use_full_doc = FALSE)
```

Arguments

<code>s</code>	A text from <code>ssd()</code> .
<code>lexicon</code>	Character vector of lexicon words, or NULL.
<code>window</code>	Context window: the number of words on each side of a lexicon word, a whole number of at least 1.
<code>sif_a</code>	SIF smoothing parameter, greater than 0. NULL weighs every word 1 (a plain mean).
<code>use_full_doc</code>	TRUE builds each document's vector from all its words, FALSE from the windows around the lexicon's words.

Value

An object of class `ssd`.

Examples

```
## Not run:
s0 <- ssd(df$essay, use_full_doc = FALSE)
lex <- suggest_lexicon(s0, "score", data = df)
s <- with_lexicon(s0, lexicon_tokens(lex)[1:10])

## End(Not run)
```

wordpack_cache	<i>The wordpack cache</i>
----------------	---------------------------

Description

Lists the packs in the directory that downloaded packs are kept in. The directory is the environment variable WORDPACK_CACHE when it is set, else `tools::R_user_dir("wordpack", "cache")`. Python ssdiff keeps its packs in `~/.wordpacks`; set `WORDPACK_CACHE=~/.wordpacks` (for example in `~/.Renvirom`) to share one copy of each pack.

Usage

```
wordpack_cache()
```

Details

A pack is about 380 MB. Nothing is deleted automatically: remove a pack you no longer need by deleting its file.

Value

The cache directory, invisibly. The packs in it and their sizes are printed.

Examples

```
wordpack_cache()
```

wordpack_lookup	<i>Rows of exact keys</i>
-----------------	---------------------------

Description

Rows of exact keys

Usage

```
wordpack_lookup(pack, keys)
```

Arguments

pack	A wordpack from wordpack_open() .
keys	Character vector of keys, matched exactly (no case change).

Value

Integer vector of 0-based rows, -1 where a key is absent.

wordpack_open	<i>Open a .wordpack file</i>
---------------	------------------------------

Description

Reads the header, the vocabulary, the character table, the stopwords, the masked-word list and the self-test. Matrix rows are read on request by [wordpack_read_rows\(\)](#).

Usage

wordpack_open(path)

Arguments

path Path to a .wordpack file.

Value

An object of class wordpack.

wordpack_read_rows	<i>Read embedding rows</i>
--------------------	----------------------------

Description

Read embedding rows

Usage

wordpack_read_rows(pack, rows, abtt = 1L)

Arguments

pack A wordpack from [wordpack_open\(\)](#).
rows 0-based rows, in the order wanted.
abtt Number of ABTT components to remove (0 to 10).

Value

A numeric matrix, one row per entry of rows.

wordpack_selftest	<i>Run the pack's self-test</i>
-------------------	---------------------------------

Description

Tokenizes the pack's stored texts and reads its stored rows, and stops at the first difference from the stored output.

Usage

```
wordpack_selftest(pack)
```

Arguments

pack	A wordpack from wordpack_open() .
------	---

Value

pack, invisibly.

wordpack_tokenize	<i>Tokenize texts with a wordpack</i>
-------------------	---------------------------------------

Description

Applies the pack's tokenizer: NFC, the fold map, emphasis stars, sentences, skipped URL and handle spans, vocabulary matches, masked words, clitics and hyphen parts.

Usage

```
wordpack_tokenize(pack, texts)
```

Arguments

pack	A wordpack from wordpack_open() .
texts	Character vector of texts.

Value

A list with

- text: the texts after NFC, fold and emphasis blanking, which every position indexes;
- sents: a data frame doc, sent, start, end, one row per sentence;
- tokens: a data frame doc, sent, key, row, start, end, one row per token, in text order.

Index

accessors, 2
AIC(), 28
BIC(), 28
cluster_snippets(accessors), 2
cluster_snippets(), 6, 26
cluster_words(accessors), 2
cluster_words(), 6
clusters(accessors), 2
clusters(), 6, 22
coef(), 17, 26
coef.ssd_pca_ols(regression-generics), 27
confint(), 17
confint.ssd_pca_ols
(regression-generics), 27
contrast, 6
contrast(), 4, 5, 7, 22, 26
df.residual(), 17
df.residual.ssd_pca_ols
(regression-generics), 27
doc_text(docs), 7
doc_text(), 5, 6
docs, 7
docs(), 5, 6
evaluate_lexicon, 8
evaluate_lexicon(), 19, 25, 30, 32
fit_groups, 9
fit_groups(), 4, 5, 7, 22, 26, 30
fit_multipls, 11
fit_multipls(), 4, 5, 7, 21, 22, 26, 30
fit_pls, 13
fit_pls(), 4, 5, 7, 16, 22, 23, 26, 30
fit_regression, 15
fit_regression(), 4, 5, 7, 14, 22, 23, 26–28, 30, 31, 33
fitted(), 17
fitted.ssd_pca_ols
(regression-generics), 27
glm(), 31
I(), 22
install_backends, 18
lexicon_tokens, 19
lexicon_tokens(), 32
lm(), 26, 31
load_wordpack, 20
load_wordpack(), 5, 6, 17, 24, 29, 30
logLik(), 17
logLik.ssd_pca_ols
(regression-generics), 27
nobs(), 17
nobs.ssd_pca_ols(regression-generics), 27
outcomes, 21
outcomes(), 5
page_data, 21
plot_sweep(sweep_table), 33
predict.ssd_pca_ols
(predict.ssd_result), 23
predict.ssd_pls(predict.ssd_result), 23
predict.ssd_result, 23
print.ssd, 24
print.ssd_contrast
(print.ssd_snippets), 25
print.ssd_embeddings, 24
print.ssd_groups(print.ssd_snippets), 25
print.ssd_lexicon, 25
print.ssd_multipls
(print.ssd_snippets), 25
print.ssd_pca_ols(print.ssd_snippets), 25

print.ssd_pls (print.ssd_snippets), 25
 print.ssd_snippets, 25
 print.wordpack, 27

 ranef(), 17
 ranef.ssd_pca_ols
 (regression-generics), 27
 readRDS(), 5
 regression-generics, 27
 residuals(), 17
 residuals.ssd_pca_ols
 (regression-generics), 27
 result-print (print.ssd_snippets), 25

 save_page (page_data), 21
 set_abtt, 29
 set_abtt(), 20
 snippets (accessors), 2
 snippets(), 6, 26
 ssd, 30
 ssd(), 8, 9, 11, 13, 15, 19, 24, 32, 34
 stats::glm(), 16, 17
 stats::lm(), 17
 stats::p.adjust(), 10
 suggest_lexicon, 31
 suggest_lexicon(), 8, 19, 25, 30, 31
 summary.ssd_contrast
 (print.ssd_snippets), 25
 summary.ssd_groups
 (print.ssd_snippets), 25
 summary.ssd_multipls
 (print.ssd_snippets), 25
 summary.ssd_pca_ols
 (print.ssd_snippets), 25
 summary.ssd_pls (print.ssd_snippets), 25
 sweep_table, 33

 utils::install.packages(), 18, 19

 vcov(), 17, 26
 vcov.ssd_pca_ols (regression-generics),
 27

 with_lexicon, 34
 with_lexicon(), 19, 30, 31
 wordpack, 20
 wordpack_cache, 35
 wordpack_cache(), 20
 wordpack_lookup, 35
 wordpack_open, 36
 wordpack_open(), 27, 35–37
 wordpack_read_rows, 36
 wordpack_read_rows(), 36
 wordpack_selftest, 37
 wordpack_tokenize, 37
 words (accessors), 2
 words(), 6, 8