Package 'RKorAPClient'

November 12, 2026

Type Package

Title 'KorAP' Web Service Client Package

Version 1.5.0

Description
A client package that makes the 'KorAP' web service API accessible from R. The corpus analysis platform 'KorAP' has been developed as a scientific tool to make potentially large, stratified and multiply annotated corpora, such as the 'German Reference Corpus DeReKo' or the 'Corpus of the Contemporary Romanian Language CoRoLa', accessible for linguists to let them verify hypotheses and to find interesting patterns in real language use. The 'RKorAPClient' package provides access to 'KorAP' and the corpora behind it for user-created R code, as a programmatic alternative to the 'KorAP' web user-interface. You can learn more about 'KorAP' and use it directly on 'DeReKo' at <https://korap.ids-mannheim.de/>.

Depends R (>= 4.1.0)

Language en-US

License BSD_2_clause + file LICENSE

URL https://github.com/KorAP/RKorAPClient/,
https://korap.ids-mannheim.de/,
https://www.ids-mannheim.de/digspra/kl/projekte/korap

BugReports https://github.com/KorAP/RKorAPClient/issues

Encoding UTF-8

Imports R.cache, broom, ggplot2, tibble, magrittr, tidyr, dplyr, lubridate, highcharter, jsonlite, keyring, utils, httr2, curl, methods, purrr, rlang, stringr, urltools, xml2

Suggests callr, chromote, lifecycle, pkgload, testthat (>= 3.1.7), tidyllm, htmlwidgets, rmarkdown, shiny, vcd, kableExtra, knitr, purrrlyr, raster, tidyverse, sf, geojsonsf, viridisLite, vcr (>= 2.0.0), webfakes, withr

Collate 'logging.R' 'KorAPConnection.R' 'KorAPCorpusStats.R' 'RKorAPClient-package.R' 'KorAPQuery.R' 'association-scores.R' 'cacheAs.R' 'chart-helpers.R' 'ci.R' 'collocationAnalysis.R' 'collocationScoreQuery.R' 'dispersion-metadata.R'

'dispersion-scores.R' 'dispersionQuery.R' 'fetchSamplePages.R'
'geom_freq_bar_ci.R' 'hc_add_onclick_korap_search.R'
'hc_dispersion.R' 'hc_freq_bar_ci.R' 'hc_freq_by_year_ci.R'
'misc.R' 'reexports.R' 'textMetadata.R'

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Marc Kupietz [aut, cre],
Nils Diewald [ctb],
Leibniz Institute for the German Language [cph, fnd]

Maintainer Marc Kupietz <kupietz@ids-mannheim.de>

Repository CRAN

Date/Publication 2026-11-12 11:10:16 UTC

Contents

association-score-functions . . . 3
auth,KorAPConnection-method . . . 5
blessCacheAs . . . 6
cacheAsInfo . . . 7
ci . . . 8
clearAccessToken,KorAPConnection-method . . . 10
clearCache,KorAPConnection-method . . . 11
collocationAnalysis,KorAPConnection-method . . . 12
collocationScoreQuery,KorAPConnection-method . . . 17
corpusPartition . . . 20
corpusQuery,KorAPConnection-method . . . 21
corpusStats,KorAPConnection-method . . . 25
dispersion-score-functions . . . 26
dispersionAnalysis,KorAPQuery-method . . . 29
dispersionQuery,KorAPConnection-method . . . 31
fetchAll,KorAPQuery-method . . . 33
fetchAnnotations . . . 34
fetchAnnotations,KorAPQuery-method . . . 35
fetchNext,KorAPQuery-method . . . 37
fetchRest,KorAPQuery-method . . . 38
fetchSamplePages,KorAPQuery-method . . . 39
frequencyQuery,KorAPConnection-method . . . 40
geom_freq_bar_ci . . . 43
hc_add_onclick_korap_search . . . 45
hc_dispersion . . . 46
hc_freq_bar_ci . . . 47
hc_freq_by_year_ci . . . 49
KorAPConnection-class . . . 51
mergeDuplicateCollocates . . . 53
persistAccessToken,KorAPConnection-method . . . 54
print.KorAPDispersion . . . 55

synsemanticStopwords 55
textMetadata,KorAPConnection-method 56
withCachedResults 57

Index **58**

`association-score-functions`
Association score functions

Description

Functions to calculate different collocation association scores between a node (target word) and words in a window around the it. The functions are primarily used by `collocationScoreQuery()`.

pmi: pointwise mutual information

mi2: pointwise mutual information squared (Daille 1994), also referred to as mutual dependency (Thanopoulos et al. 2002)

mi3: pointwise mutual information cubed (Daille 1994), also referred to as log-frequency biased mutual dependency) (Thanopoulos et al. 2002)

logDice: log-Dice coefficient, a heuristic measure that is popular in lexicography (Rychlý 2008)

ll: log-likelihood (Dunning 1993) using Stefan Evert's (2004) simplified implementation

Usage

```
defaultAssociationScoreFunctions()

pmi(O1, O2, O, N, E, window_size)

mi2(O1, O2, O, N, E, window_size)

mi3(O1, O2, O, N, E, window_size)

logDice(O1, O2, O, N, E, window_size)

ll(O1, O2, O, N, E, window_size)
```

Arguments

`O1`	observed absolute frequency of node
`O2`	observed absolute frequency of collocate
`O`	observed absolute frequency of collocation
`N`	corpus size
`E`	expected absolute frequency of collocation (already adjusted to window size)
`window_size`	total window size around node (left neighbour count + right neighbour count)

Details

logDice is computed as defined by Rychlý (2008), that is from the plain marginal frequencies of node and collocate, so that its values are comparable to those of other tools, such as Sketch Engine. Unlike the expectation based scores, it does not take the window size into account: the Dice coefficient relates the co-occurrence frequency to how often the two words occur at all, and 01 and 02 count word tokens, while a window size factor would count window positions.

Value

```
association score
```

Salience versus surprise

logDice is the only one of these scores that does not compare the observed co-occurrence frequency to an expected one. As it merely puts 0 into the numerator of the Dice coefficient, the difference between the score of a pair and the score that same pair would get if it co-occurred exactly as often as expected is precisely the pointwise mutual information:

`logDice(O) - logDice(E) = log2(O / E) = pmi`

The level a pair starts from therefore depends on the marginal frequencies alone, and it is high whenever both words are frequent. Collocates of *Grund* in DeReKo, in a window of five words to each side, illustrate this:

- *triftiger* is rare, so it starts from a logDice of -4.60 and reaches 3.96, exceeding what is expected by a pmi of 8.57
- *Berlin* is frequent, so it starts from 5.58 and still reaches 3.84, while co-occurring 1.74 bits *less* often than expected

A frequent collocate can thus reach a respectable logDice although the node does not attract it at all. logDice measures how salient a pair is, given how often its words occur, rather than how surprising it is. That is what it was designed for (Rychlý 2008), and it is why its values do not depend on the corpus size and are comparable across corpora.

Since `collocationAnalysis()` ranks and thresholds by logDice, it therefore drops collocates occurring less often than expected by default. Its `minObservedExpectedRatio` parameter controls this: raise it to demand a stronger contrast, or set it to 0 to see the unfiltered ranking, for instance to study repulsion. `collocationScoreQuery()` does not filter, as there the pairs to score are given explicitly. For results obtained otherwise, `dplyr::filter(O > E)` or a minimum pmi or ll has the same effect.

References

Daille, B. (1994): Approche mixte pour l'extraction automatique de terminologie: statistiques lexicales et filtres linguistiques. PhD thesis, Université Paris 7.

Thanopoulos, A., Fakotakis, N., Kokkinakis, G. (2002): Comparative evaluation of collocation extraction metrics. In: Proc. of LREC 2002: 620–625.

Rychlý, Pavel (2008): A lexicographer-friendly association score. In Proceedings of Recent Advances in Slavonic Natural Language Processing, RASLAN, 6–9. https://www.fi.muni.cz/usr/sojka/download/raslan2008/13.pdf.

Dunning, T. (1993): Accurate methods for the statistics of surprise and coincidence. Comput. Linguist. 19, 1 (March 1993), 61-74.

Evert, Stefan (2004): The Statistics of Word Cooccurrences: Word Pairs and Collocations. PhD dissertation, IMS, University of Stuttgart. Published in 2005, URN urn:nbn:de:bsz:93-opus-23714. Free PDF available from https://purl.org/stefan.evert/PUB/Evert2004phd.pdf

See Also

Other collocation analysis functions: collocationAnalysis,KorAPConnection-method, collocationScoreQuery,KorAP
synsemanticStopwords()

Examples

```
## Not run:

KorAPConnection(verbose = TRUE) %>%
collocationScoreQuery("Perlen", c("verziertes", "Säue"),
  scoreFunctions = append(defaultAssociationScoreFunctions(),
     list(localMI = function(O1, O2, O, N, E, window_size) {
                       O * log2(O/E)
                     })))

## End(Not run)
```

---

auth,KorAPConnection-method

Authorize RKorAPClient

Description

Authorize RKorAPClient to make KorAP queries and download results on behalf of the user.

Usage

```
## S4 method for signature 'KorAPConnection'
auth(
  kco,
  app_id = generic_kor_app_id,
  app_secret = NULL,
  scope = kco@oauthScope
)
```

Arguments

kco	KorAPConnection object
app_id	OAuth2 application id. Defaults to the generic KorAP client application id.

app_secret	OAuth2 application secret. Used with confidential client applications. Defaults to NULL.
scope	OAuth2 scope. Defaults to "search match_info".

Value

KorAPConnection object with access token set in @accessToken.

See Also

persistAccessToken(), clearAccessToken()

Other initialization functions: KorAPConnection-class, clearAccessToken,KorAPConnection-method, persistAccessToken,KorAPConnection-method

Examples

```
## Not run:
kco <- KorAPConnection(verbose = TRUE) %>% auth()
df <- collocationAnalysis(kco, "focus([marmot/p=ADJA] {Ameisenplage})",
  leftContextSize = 1, rightContextSize = 0
)

## End(Not run)
```

---

blessCacheAs *Vouch for a cacheAs file that an older version wrote*

Description

Association scores changed in 1.4.0, so files from before it are recomputed rather than used (see cacheAs). Where a file is known to hold what this version would compute - because it was written by a development version that already had the corrections, for instance - this records that, and the file is used again as it is.

Usage

```
blessCacheAs(cacheAs)
```

Arguments

cacheAs	paths of the files to vouch for, with or without their .rds extension

Details

What actually wrote a file is left as it stands; the blessing is recorded beside it, so that cacheAsInfo() keeps telling the truth about where the numbers come from.

A file that records no parameters, as those written before 1.3.0.9000 do, has nothing left to be compared against a call once it is blessed, and is therefore reused for any call that names it. Bless such a file only if that is what you mean.

Value

the paths, invisibly

See Also

Other cacheAs: `cacheAsInfo()`, `withCachedResults()`

Examples

```
## Not run:
blessCacheAs("klima-ca.rds")
blessCacheAs(list.files("data", pattern = "\\.rds$", full.names = TRUE))

## End(Not run)
```

`cacheAsInfo` *What produced a cacheAs file*

Description

Reads back what a query function recorded in a cacheAs file: the parameters it was called with, the KorAP instance it asked, the index revision that instance's corpus had at the time, and the version of RKorAPClient that wrote the file. Useful for saying, of a result kept next to a document, what the numbers in it rest on.

Usage

```
cacheAsInfo(cacheAs)
```

Arguments

`cacheAs`	path to the file, with or without its `.rds` extension

Value

a list with the elements `scoreVersion`, `packageVersion`, `parameters`, `dots`, `ignoredParameters` (inactive options, when recorded), `apiUrl` and `indexRevision`, or `NULL` for a file written by a version before 1.4.0, which recorded none of this

See Also

Other cacheAs: `blessCacheAs()`, `withCachedResults()`

Examples

```
## Not run:
KorAPConnection() |> frequencyQuery("Ameisenplage", cacheAs = "ameisenplage.rds")
cacheAsInfo("ameisenplage.rds")

## End(Not run)
```

---

ci *Add confidence interval and relative frequency variables*

Description

Using `prop.test()`, `ci` adds three columns to a data frame:

1. relative frequency (`f`)
2. lower bound of a confidence interval (`ci.low`)
3. upper bound of a confidence interval

Convenience function for converting frequency tables to instances per million.

Convenience function for converting frequency tables of alternative variants (generated with `as.alternatives=TRUE`) to percent.

Converts a vector of query or vc strings to typically appropriate legend labels by clipping off prefixes and suffixes that are common to all query strings.

Experimental convenience function for plotting typical frequency by year graphs with confidence intervals using ggplot2. **Warning:** This function may be moved to a new package.

Usage

```
ci(df, x = totalResults, N = total, conf.level = 0.95)

ipm(df)

percent(df)

queryStringToLabel(data, pubDateOnly = FALSE, excludePubDate = FALSE)

geom_freq_by_year_ci(
  mapping = aes(ymin = conf.low, ymax = conf.high),
  showCI = TRUE,
  smooth = FALSE,
  ...
)
```

Arguments

df	table returned from frequencyQuery()
x	column with the observed absolute frequency.
N	column with the total frequencies
conf.level	confidence level of the returned confidence interval. Must be a single number between 0 and 1.
data	string or vector of query or vc definition strings
pubDateOnly	discard all but the publication date
excludePubDate	discard publication date constraints
mapping	Set of aesthetic mappings created by aes() or aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
showCI	Show confidence bands (default TRUE). FALSE retains only the frequency curves and points and does not require interval bounds.
smooth	Draw curved connections between annual observations (default FALSE). Uses natural cubic spline interpolation within each group and facet, on the plot's transformed scales. Missing values and missing calendar years break the curves. Points remain at the observed years.
...	Other arguments passed to geom_ribbon, geom_line, and geom_click_point.

Details

Given a table with columns f, conf.low, and conf.high, ipm ads a column ipm und multiplies conf.low and conf.high with 10^6.

With smooth = TRUE, map numeric calendar years to x and frequency to y. Interpolation is only a display aid, not a fitted trend or a trend test. Confidence-band boundaries are interpolated between the supplied annual limits; they are not confidence intervals for unobserved dates. Observed annual values and limits are retained. As with other ggplot2 statistics, interpolation follows the axis scales, including logarithmic scales. Cubic curves can overshoot between annual values; use straight connections when the exact shape matters. Negative interpolated frequencies are clipped at zero (on a log axis, zero is outside the displayed range). Each group needs at most one observation per year; use colour, fill or group to distinguish expressions or corpora. No extrapolation is performed. The interpolation used by Highcharts may give a different curve between the same annual observations.

Value

original table with additional column ipm and converted columns conf.low and conf.high

original table with converted columns f, conf.low and conf.high

string or vector of strings with clipped off common prefixes and suffixes

See Also

ci is already included in frequencyQuery()

Examples

```
## Not run:

library(ggplot2)
kco <- KorAPConnection(verbose=TRUE)
expand_grid(year=2015:2018, alternatives=c("Hate Speech", "Hatespeech")) %>%
  bind_cols(corpusQuery(kco, .$alternatives, sprintf("pubDate in %d", .$year))) %>%
  mutate(total=corpusStats(kco, vc=vc)$tokens) %>%
  ci() %>%
  ggplot(aes(x=year, y=f, fill=query, color=query, ymin=conf.low, ymax=conf.high)) +
    geom_point() + geom_line() + geom_ribbon(alpha=.3)

## End(Not run)
## Not run:

KorAPConnection() %>% frequencyQuery("Test", paste0("pubDate in ", 2000:2002)) %>% ipm()

## End(Not run)
## Not run:

KorAPConnection() %>%
    frequencyQuery(c("Tollpatsch", "Tolpatsch"),
    vc=paste0("pubDate in ", 2000:2002),
    as.alternatives = TRUE) %>%
  percent()

## End(Not run)
queryStringToLabel(paste("textType = /Zeit.*/ & pubDate in", c(2010:2019)))
queryStringToLabel(c("[marmot/m=mood:subj]", "[marmot/m=mood:ind]"))
queryStringToLabel(c("wegen dem [tt/p=NN]", "wegen des [tt/p=NN]"))

## Not run:
library(ggplot2)
kco <- KorAPConnection(verbose=TRUE)

expand_grid(condition = c("idsColumn = /Wirtschaft.*/", "idsColumn != /Wirtschaft.*/"),
            year = (2005:2011)) %>%
  cbind(frequencyQuery(kco, "[tt/l=Heuschrecke]",
                           paste0(.$condition," & pubDate in ", .$year)))  %>%
  ipm() %>%
  ggplot(aes(year, ipm, fill = condition, color = condition)) +
  geom_freq_by_year_ci()

## End(Not run)
```

`clearAccessToken,KorAPConnection-method`

Clear access token from keyring and KorAPConnection object

Description

Clear access token from keyring and KorAPConnection object

Usage

```
## S4 method for signature 'KorAPConnection'
clearAccessToken(kco)
```

Arguments

kco	KorAPConnection object

Value

KorAPConnection object with access token set to NULL.

See Also

persistAccessToken()

Other initialization functions: KorAPConnection-class, auth,KorAPConnection-method, persistAccessToken,KorAPC

Examples

```
## Not run:
kco <- KorAPConnection()
kco <- clearAccessToken(kco)

## End(Not run)
```

---

clearCache,KorAPConnection-method

Clear local cache

Description

Clears the local cache of API responses for the current RKorAPClient version. Useful when you want to force fresh data retrieval or free up disk space.

Usage

```
## S4 method for signature 'KorAPConnection'
clearCache(kco)
```

Arguments

kco	KorAPConnection object

Value

Invisible NULL (function called for side effects)

Examples

```
## Not run:
kco <- KorAPConnection()
clearCache(kco)

## End(Not run)
```

---

collocationAnalysis,KorAPConnection-method
Collocation analysis

Description

Performs a collocation analysis for the given node (or query) in the given virtual corpus.

Usage

```
## S4 method for signature 'KorAPConnection'
collocationAnalysis(
  kco,
  node,
  vc = "",
  lemmatizeNodeQuery = FALSE,
  minOccur = 5,
  leftContextSize = 5,
  rightContextSize = 5,
  topCollocatesLimit = 200,
  searchHitsSampleLimit = 20000,
  ignoreCollocateCase = FALSE,
  withinSpan = ifelse(exactFrequencies, "base/s=s", ""),
  exactFrequencies = TRUE,
  stopwords = append(RKorAPClient::synsemanticStopwords(), node),
  seed = 7,
  expand = length(vc) != length(node),
  maxRecurse = 0,
  addExamples = FALSE,
  thresholdScore = "logDice",
  threshold = 2,
  localStopwords = c(),
  collocateFilterRegex = "^[:alnum:]+-?[:alnum:]*$",
  minObservedExpectedRatio = 1,
  queryMissingScores = FALSE,
```

```
  missingScoreQuantile = 0.05,
  vcLabel = NA_character_,
  cacheAs = NULL,
  ...
)
```

Arguments

kco	KorAPConnection() object (obtained e.g. from KorAPConnection()
node	target word or query as a single character string
vc	character vector describing the virtual corpus or corpora in which the query should be performed. An empty string (default) means the whole corpus, as far as it is license-wise accessible.
lemmatizeNodeQuery	if TRUE, node query will be lemmatized, i.e. x -> [tt/l=x]
minOccur	minimum absolute number of observed co-occurrences to consider a collocate candidate
leftContextSize	size of the left context window
rightContextSize	size of the right context window
topCollocatesLimit	limit analysis to the n most frequent collocates in the search hits sample
searchHitsSampleLimit	limit the size of the search hits sample
ignoreCollocateCase	logical, set to TRUE if collocate case should be ignored
withinSpan	KorAP span specification (see https://korap.ids-mannheim.de/doc/ql/poliqarp-plus?embedded=true#spans) for collocations to be searched within. Defaults to base/s=s.
exactFrequencies	if FALSE, extrapolate observed co-occurrence frequencies from frequencies in search hits sample, otherwise retrieve exact co-occurrence frequencies
stopwords	vector of stopwords not to be considered as collocates
seed	seed for random page collecting order
expand	if TRUE, node and vc parameters are expanded to all of their combinations
maxRecurse	apply collocation analysis recursively maxRecurse times
addExamples	If TRUE, examples for instances of collocations will be added in a column example. This makes a difference in particular if node is given as a lemma query.
thresholdScore	association score function (see association-score-functions) to use for computing the threshold that is applied for recursive collocation analysis calls (only applied when maxRecurse > 0)

`threshold` minimum value of `thresholdScore` function call to apply collocation analysis recursively (only applied when `maxRecurse > 0`). Note that the default score, `logDice`, expresses how salient a pair is rather than how surprising, so that a frequent collocate can pass it while co-occurring less often than expected. `minObservedExpectedRatio` keeps those out. See the "Salience versus surprise" section of `association-score-functions`.

`localStopwords` vector of stopwords that will not be considered as collocates in the current function call, but that will not be passed to recursive calls

`collocateFilterRegex` allow only collocates matching the regular expression

`minObservedExpectedRatio` minimum ratio of observed to expected co-occurrence frequency a collocate must reach. Defaults to 1, which keeps only collocates that occur at least as often as expected by chance, corresponding to a non-negative `pmi`. Without it, frequent words can end up among the top collocates by `logDice` although the node does not attract them at all (see the "Salience versus surprise" section of `association-score-functions`). Raise it to demand a stronger contrast, e.g. 2 for collocates occurring at least twice as often as expected, or set it to 0 to switch the filter off and obtain the unfiltered result of earlier versions, e.g. in order to study repulsion.

`queryMissingScores` if TRUE, attempt to retrieve corpus-based association scores for vc/collocate combinations that would otherwise be imputed, by re-querying the KorAP backend without applying the collocate frequency threshold

`missingScoreQuantile` lower quantile (evaluated per association measure over the pooled result set) that anchors the adaptive floor used for imputing missing scores between virtual corpora; a robust spread is subtracted from this anchor so the imputed values stay at or below the weakest observed scores. Imputed cells are marked in the `imputed*` columns; see the section on interpreting multi-VC comparisons below

`vcLabel` optional label override for the current virtual corpus (used internally when named VC collections are expanded)

`cacheAs` path to an RDS file to keep the result in. If the file exists and records the same call, it is read back instead of contacting the server; otherwise the query is run and its result stored there. Unlike the connection's `cache`, this file belongs to the caller, which is what keeps an analysis reproducible once the corpus has grown or the scores have changed. Defaults to `NULL` (no file).

The analysis parameters are stored alongside the result. If they differ from those of the current call, the cached result would not be the one that was asked for, so it is recomputed and the file overwritten, with a warning naming the parameters that differ. Pass a different `cacheAs` file name to keep an existing analysis. Cache files written by RKorAPClient 1.3.0 do not contain the parameters yet and are used as they are.

`...` more arguments will be passed to `collocationScoreQuery()`

Details

The collocation analysis is currently implemented on the client side, as some of the functionality is not yet provided by the KorAP backend. Mainly for this reason it is very slow (several minutes, up to hours), but on the other hand very flexible. You can, for example, perform the analysis in arbitrary virtual corpora, use complex node queries, and look for expression-internal collocates using the focus function (see examples and demo).

To increase speed at the cost of accuracy and possible false negatives, you can decrease searchHitsSampleLimit and/or topCollocatesLimit and/or set exactFrequencies to FALSE.

Note that some outdated non-DeReKo back-ends might not yet support returning tokenized matches (warning issued). In this case, the client library will fall back to client-side tokenization which might be slightly less accurate. This might lead to false negatives and to frequencies that differ from corresponding ones acquired via the web user interface.

Value

A tibble where each row represents a candidate collocate for the requested node. Columns include (depending on the selected association measures):

- `node`, `collocate`, `vc`, `label`: identifiers for the query node, collocate, virtual corpus, and optional label.
- Frequency and contingency information such as `frequency`, `O`, `O1`, `O2`, `E`, `leftContextSize`, `rightContextSize`, and `w`.
- Association measures (e.g. `logDice`, `ll`, `mi`, ...), one column per requested scorer.
- Per-labelled association scores produced by multi-VC comparisons using the pattern `<measure>_<label>`.
- Ranks per label/measure with the pattern `rank_<label>_<measure>` (1 is best) and the corresponding percentile ranks `percentile_rank_<label>_<measure>`.
- Pairwise contrasts for two-label comparisons, e.g. `delta_<measure>`, `delta_rank_<measure>`, and `delta_percentile_rank_<measure>`.
- Summary columns describing the strongest labels per measure (`winner_*`, `runner_up_*`, `loser_*`, and `max_delta_*`), including winner/loser `webUIRequestUrl` columns. In multi-VC comparisons, missing per-label concordance URLs are derived from another available row URL for the same `node`/`collocate` by replacing the `cq` parameter with the target label's virtual corpus. Unsuffixed `winner_webUIRequestUrl` and `loser_webUIRequestUrl` columns are populated only when the score-based URL choices agree.
- `imputed_<label>`: whether the score for that label was imputed rather than observed (see `missingScoreQuantile`). `n_imputed` counts them, and `imputed` is `n_imputed > 0`: it describes the node/collocate pair across all labels, not the label of the row it stands in. A row can therefore carry `imputed = TRUE` while its own scores are perfectly attested, because the pair is missing from some other virtual corpus - use `imputed_<label>` for the row itself. Filter with `dplyr::filter(!imputed)` to keep only collocates attested in every compared virtual corpus.
- Optional helper columns such as `query`, `example`, or `url` when example retrieval is requested.

Interpreting multi-VC comparisons

[Experimental]

The comparison columns produced when `vc` holds more than one virtual corpus are experimental: their names and semantics may still change in a future release without a deprecation cycle. Code that has to keep working across versions should select the columns it needs explicitly.

They are an exploration aid, not a significance test. When reading them, keep three properties in mind.

Imputed scores describe presence/absence, not contrast. A collocate that passes the `minOccur` and `topCollocatesLimit` thresholds in one virtual corpus but not in another has no observed score for the latter. Such cells are imputed from a floor derived from the pooled result set (see `missingScoreQuantile`), so the corresponding `delta_*` and `max_delta_*` values measure the distance to that floor rather than an attested difference. The `imputed`, `n_imputed` and `imputed_<label>` columns mark these rows; `dplyr::filter(!imputed)` restricts the result to collocates attested everywhere, and `queryMissingScores = TRUE` replaces most imputed cells with scores actually retrieved from the backend. Mind what `imputed` is about: the pair, not the row. It is `TRUE` as soon as one label lacks the collocate, and stays `TRUE` on the rows of the labels where it is attested, which is what makes `dplyr::filter(!imputed)` drop the pair as a whole. Whether the row at hand rests on an imputed score is what `imputed_<label>` says.

Per-label columns carry syntactic names. The label in `<measure>_<label>`, `rank_<label>_<measure>` and `imputed_<label>` is the one the caller gave, put through `make.names()`, so that the result stays a well formed data frame: a virtual corpus named `1976-1980` appears as `logDice_X1976.1980`. The `label` column and the `winner_*` / `loser_*` columns keep the name as it was given, so mapping between the two means applying the same transformation, e.g. `stats::setNames(make.names(labels), labels)`.

Imputed values are relative to one analysis. The floor is computed from the scores present in the result at hand. Analysing a node on its own and analysing it together with other nodes therefore yield different imputed values, and deltas involving imputed cells are not comparable across separate calls. Deltas between observed scores are unaffected.

Winners carry no uncertainty. Unlike `ci()`, which attaches confidence intervals to relative frequencies, the `winner_*` / `loser_*` columns simply order point estimates. A collocate wins by a hair on six occurrences exactly as decisively as one that wins by a wide margin on thousands. Consult the observed frequencies (`O`, `O1`, `O2`) and the `webUIRequestUrl` concordance links before drawing conclusions from a small difference.

Note also that `rank_<label>_<measure>` and `percentile_rank_<label>_<measure>` are computed within each label, over that label's own candidate set. Candidate sets usually differ in size between virtual corpora, so rank-based deltas compare positions in populations of different sizes.

Slow queries

Complex queries can take minutes on large corpora, especially on DeLiKo@DNB (`https://korap.dnb.de`), or exceed the time limit of the server, which leaves the results incomplete. As long as KorAP does not offer sampling by itself, a workaround is to restrict `vc` to a pseudo-random sample of texts, selected by the last digit of their sigle: `"textSigle = /.*5/"` selects about a tenth, `"textSigle = /.*[0-4]/"` about half of the texts. Combine the sample with other conditions using `&`, e.g. `"textType = /.*Krimi.*/ & textSigle = /.*5/"`. Relative frequencies, and association scores that do not depend on the corpus size, such as logDice, remain estimates for the whole (virtual)

corpus, whereas absolute frequencies, and with them log-likelihood scores, shrink with the sample. A sample saves about as much time as it leaves out if the server has recently read the index of the query terms, e.g. for the same query on another virtual corpus, but hardly any time otherwise, when most of the time goes into reading the index from disk. It thus pays off in series of queries on the same terms, such as a `frequencyQuery()` over many years or a `collocationAnalysis()`. Repeating a query on the same `vc` takes no time anyway, as its result comes from the local cache (see the `cache` parameter of `KorAPConnection()`). Queries that are fast anyway do not get faster, because selecting the sample takes some time itself.

See Also

Other collocation analysis functions: `association-score-functions`, `collocationScoreQuery,KorAPConnection-meth`
`synsemanticStopwords()`

Examples

```
## Not run:

# Find top collocates of "Packung" inside and outside the sports domain.
KorAPConnection(verbose = TRUE) |> auth() |>
  collocationAnalysis("Packung",
    vc = c("dmozDomain = /sport.*/", "dmozDomain != /sport.*/"),
    leftContextSize = 1, rightContextSize = 1, topCollocatesLimit = 20
  ) |>
  dplyr::filter(logDice >= 5)

## End(Not run)

## Not run:

# Identify the most prominent light verb construction with "in ... setzen".
# Note that, currently, the use of focus function disallows exactFrequencies.
KorAPConnection(verbose = TRUE) |>
  collocationAnalysis("focus(in [tt/p=NN] {[tt/l=setzen]})",
   leftContextSize = 1, rightContextSize = 0, exactFrequencies = FALSE, topCollocatesLimit = 20
  )

## End(Not run)
```

---

collocationScoreQuery,KorAPConnection-method

Query frequencies of a node and a collocate and calculate collocation association scores

---

Description

Computes various collocation association scores based on `frequencyQuery()`s for a target word and a collocate.

Usage

```
## S4 method for signature 'KorAPConnection'
collocationScoreQuery(
  kco,
  node,
  collocate,
  vc = "",
  lemmatizeNodeQuery = FALSE,
  lemmatizeCollocateQuery = FALSE,
  leftContextSize = 5,
  rightContextSize = 5,
  scoreFunctions = defaultAssociationScoreFunctions(),
  smoothingConstant = 0.5,
  observed = NA,
  ignoreCollocateCase = FALSE,
  withinSpan = "base/s=s",
  cacheAs = NULL
)
```

Arguments

kco	KorAPConnection() object (obtained e.g. from KorAPConnection()
node	target word or query as a single character string
collocate	character vector of one or more collocates of the target word
vc	character vector describing the virtual corpus or corpora in which the query should be performed. An empty string (default) means the whole corpus, as far as it is license-wise accessible.
lemmatizeNodeQuery	logical, set to TRUE if node query should be lemmatized, i.e. x -> [tt/l=x]
lemmatizeCollocateQuery	logical, set to TRUE if collocate query should be lemmatized, i.e. x -> [tt/l=x]
leftContextSize	size of the left context window
rightContextSize	size of the right context window
scoreFunctions	named list of score functions of the form function(O1, O2, O, N, E, window_size), see e.g. pmi
smoothingConstant	smoothing constant will be added to all observed values
observed	if collocation frequencies are already known (or estimated from a sample) they can be passed as a vector here, otherwise: NA
ignoreCollocateCase	logical, set to TRUE if collocate case should be ignored
withinSpan	KorAP span specification (see https://korap.ids-mannheim.de/doc/ql/poliqarp-plus?embedded=true#spans) for collocations to be searched within. Defaults to base/s=s.

cacheAs — path to an RDS file to keep the result in. If the file exists and records the same call, it is read back instead of contacting the server; otherwise the query is run and its result stored there. Unlike the connection's cache, this file belongs to the caller, which is what keeps an analysis reproducible once the corpus has grown or the scores have changed. Defaults to NULL (no file).

Value

tibble with query KorAP web request URL, all observed values and association scores

See Also

Other collocation analysis functions: association-score-functions, collocationAnalysis,KorAPConnection-method, synsemanticStopwords()

Examples

```
## Not run:

KorAPConnection(verbose = TRUE) |>
  collocationScoreQuery("Grund", "triftiger")

## End(Not run)

## Not run:

KorAPConnection(verbose = TRUE) |>
collocationScoreQuery("Grund", c("guter", "triftiger"),
  scoreFunctions = list(localMI = function(O1, O2, O, N, E, window_size) { O * log2(O/E) }) )

## End(Not run)

## Not run:

library(highcharter)
library(tidyr)
KorAPConnection(verbose = TRUE) |>
  collocationScoreQuery("Team", "agil", vc = paste("pubDate in", c(2014:2018)),
                        lemmatizeNodeQuery = TRUE, lemmatizeCollocateQuery = TRUE) |>
                pivot_longer(14:last_col(), names_to = "measure", values_to = "score") |>
  hchart(type="spline", hcaes(label, score, group=measure)) |>
  hc_add_onclick_korap_search()

## End(Not run)
```

corpusPartition *Define corpus parts for a dispersion analysis*

Description

Discover metadata groups and their token counts, or define a custom partition. Discovery checks the base corpus independently of query hits, so that parts without hits are retained without downloading the corpus.

Usage

```
corpusPartition(
  kco,
  by,
  values = NULL,
  vc = "",
  parts = NULL,
  classify = NULL,
  fields = NULL,
  verbose = kco@verbose,
  cacheAs = NULL,
  regex = NULL
)
```

Arguments

kco	A KorAPConnection() object.
by	Metadata field. The aliases source, year and region stand for corpusTitle, pubDate (publication year) and pubPlaceKey respectively.
values	Vector of metadata values; optional names label the parts. If neither values nor parts is supplied, discover the groups automatically. Four-digit values for pubDate or creationDate define years. Other fields use literal equality. Use the stored region keys, not place names. Corpus titles may include publication years. To group annual corpora by source, use by = "source", as in demo("dispersion"). Custom parts and a matching classify function support other groupings.
vc	Base virtual corpus. All parts are intersected with this corpus.
parts	Alternatively, a uniquely named vector of virtual-corpus definitions. For sampling arbitrary definitions, also provide classify.
classify	Optional function of a matches data frame returning one part label per row, or NA for excluded/missing metadata. Its membership rules must agree with parts. Multivalued fields require an explicit classifier.
fields	Metadata fields needed by classify; defaults to by.
verbose	Whether to print progress.
cacheAs	Optional result file; see frequencyQuery().

`regex` Optional trailing-suffix grouping rule for automatic discovery; see dispersionAnalysis,KorAPQuery-method. Defaults to removing a year suffix for `by = "source"`.

Value

A tibble of class `KorAPCorpusPartition` with `part`, `value`, `vc`, and `tokens` columns. Attributes record the metadata mapping, base corpus, server, index revision, covered tokens and excluded tokens. Overlapping parts are rejected by comparing their summed sizes with their union. An incomplete inventory explicitly restricts the population to its union, with a warning; it does not silently stand for the whole base corpus.

Examples

```
## Not run:
kco <- KorAPConnection()
years <- corpusPartition(kco, "year", 2020:2022, vc = "referTo ratskorpus")
countries <- corpusPartition(kco, "region", c("DE", "AT", "CH"))
dispersionQuery(kco, "NutzerInnen", partitions = list(year = years))

## End(Not run)
```

---

corpusQuery,KorAPConnection-method

Search corpus for query terms

Description

`corpusQuery` performs a corpus query via a connection to a KorAP-API-server

Usage

```
## S4 method for signature 'KorAPConnection'
corpusQuery(
  kco,
  query = if (missing(KorAPUrl)) {

   stop("At least one of the parameters query and KorAPUrl must be specified.", call. =
     FALSE)
 } else {
     httr2::url_parse(KorAPUrl)$query$q
 },
  vc = if (missing(KorAPUrl)) "" else httr2::url_parse(KorAPUrl)$query$cq,
  KorAPUrl,
  metadataOnly = TRUE,
 ql = if (missing(KorAPUrl)) "poliqarp" else httr2::url_parse(KorAPUrl)$query$ql,
  fields = c("corpusSigle", "corpusTitle", "textSigle", "pubDate", "pubPlace",
   "availability", "textClass", "dmozDomain", "wikiDomain", "snippet", "tokens"),
  accessRewriteFatal = TRUE,
```

```
  verbose = kco@verbose,
  expand = length(vc) != length(query),
  as.df = FALSE,
  context = NULL
)
```

Arguments

kco	KorAPConnection() object (obtained e.g. from KorAPConnection())
query	string that contains the corpus query. The query language depends on the ql parameter. Either query must be provided or KorAPUrl.
vc	string describing the virtual corpus in which the query should be performed. An empty string (default) means the whole corpus, as far as it is license-wise accessible.
KorAPUrl	instead of providing the query and vc string parameters, you can also simply copy a KorAP query URL from your browser and use it here (and in KorAPConnection) to provide all necessary information for the query.
metadataOnly	logical that determines whether queries should return only metadata without any snippets. This can also be useful to prevent access rewrites. Note that the default value is TRUE. If you want your corpus queries to return not only metadata, but also KWICs, you need to authorize your RKorAPClient application as explained in the authorization section of the RKorAPClient Readme on GitHub and set the metadataOnly parameter to FALSE.
ql	string to choose the query language (see section on Query Parameters in the Kustvakt-Wiki for possible values).
fields	character vector specifying which metadata fields to retrieve for each match. Available fields depend on the corpus. For DeReKo (German Reference Corpus), possible fields include: **Text identification:** textSigle, docSigle, corpusSigle - hierarchical text identifiers **Publication info:** author, editor, title, docTitle, corpusTitle - authorship and titles **Temporal data:** pubDate, creationDate - when text was published/created **Publication details:** pubPlace, publisher, reference - where/how published **Text classification:** textClass, dmozDomain, wikiDomain, textType, textTypeArt, idsColumn - topic domain, genre, text type and column **Administrative and technical info:** corpusEditor, availability, language, foundries - access rights and annotations **Content data:** snippet, tokens, tokenSource, externalLink - actual text content, tokenization, and link to source text **System data:** indexCreationDate, indexLastModified - corpus indexing info Current DeReKo releases use dmozDomain and wikiDomain for topic domains; textClass is retained in the default request for older corpus releases. Use c("textSigle", "pubDate", "author") to retrieve multiple fields. Default fields provide basic text identification and publication metadata. The actual text

content fields (snippet and tokens) are activated by default if metadataOnly is set to FALSE.

accessRewriteFatal
: abort if query or given vc had to be rewritten due to insufficient rights (not yet implemented).

verbose
: print some info

expand
: logical that decides if query and vc parameters are expanded to all of their combinations. Defaults to TRUE, iff query and vc have different lengths

as.df
: return result as data frame instead of as S4 object?

context
: string that specifies the size of the left and the right context returned in snippet (provided that metadataOnly is set to false and that the necessary access rights are met). The format of the context size specification (e.g. 3-token,3-token) is described in the Service: Search GET documentation of the Kustvakt Wiki. If the parameter is not set, the default context size specification of the KorAP server instance will be used. Note that you cannot overrule the maximum context size set in the KorAP server instance, as this is typically legally motivated.

Value

Depending on the as.df parameter, a tibble or a KorAPQuery() object that, among other information, contains the total number of results in @totalResults. Multiple queries return a summary table. With as.df = FALSE, it also retains query states for fetchAll(), fetchNext(), fetchSamplePages() and dispersionAnalysis(). Names of the query vector label the comparison columns. With as.df = TRUE, only the summary is retained. A corresponding URL to be used within a web browser is contained in @webUIRequestUrl Please make sure to check $collection$rewrites to see if any unforeseen access rewrites of the query's virtual corpus had to be performed.

Slow queries

Complex queries can take minutes on large corpora, especially on DeLiKo@DNB (https://korap.dnb.de), or exceed the time limit of the server, which leaves the results incomplete. As long as KorAP does not offer sampling by itself, a workaround is to restrict vc to a pseudo-random sample of texts, selected by the last digit of their sigle: "textSigle = /.*5/" selects about a tenth, "textSigle = /.*[0-4]/" about half of the texts. Combine the sample with other conditions using &, e.g. "textType = /.*Krimi.*/ & textSigle = /.*5/". Relative frequencies, and association scores that do not depend on the corpus size, such as logDice, remain estimates for the whole (virtual) corpus, whereas absolute frequencies, and with them log-likelihood scores, shrink with the sample. A sample saves about as much time as it leaves out if the server has recently read the index of the query terms, e.g. for the same query on another virtual corpus, but hardly any time otherwise, when most of the time goes into reading the index from disk. It thus pays off in series of queries on the same terms, such as a frequencyQuery() over many years or a collocationAnalysis(). Repeating a query on the same vc takes no time anyway, as its result comes from the local cache (see the cache parameter of KorAPConnection()). Queries that are fast anyway do not get faster, because selecting the sample takes some time itself.

References

https://ids-pub.bsz-bw.de/frontdoor/index/index/docId/9026

See Also

KorAPConnection(), fetchNext(), fetchRest(), fetchAll(), corpusStats()

Other corpus search functions: fetchAll,KorAPQuery-method, fetchAnnotations,KorAPQuery-method, fetchNext,KorAPQuery-method

Examples

```
## Not run:

# Fetch basic metadata for "Ameisenplage"
KorAPConnection() |>
  corpusQuery("Ameisenplage") |>
  fetchAll()

# Fetch specific metadata fields for bibliographic analysis
query <- KorAPConnection() |>
  corpusQuery("Ameisenplage",
              fields = c("textSigle", "author", "title", "pubDate", "pubPlace", "textType"))
results <- fetchAll(query)
results@collectedMatches

## End(Not run)

## Not run:

# Use the copy of a KorAP-web-frontend URL for an API query of "Ameise" in a virtual corpus
# and show the number of query hits (but don't fetch them).

KorAPConnection(verbose = TRUE) |>
  corpusQuery(
    KorAPUrl =
      "https://korap.ids-mannheim.de/?q=Ameise&cq=pubDate+since+2017&ql=poliqarp"
  )

## End(Not run)

## Not run:

# Plot the time/frequency curve of "Ameisenplage"
KorAPConnection(verbose = TRUE) |>
  {
    . ->> kco
  } |>
  corpusQuery("Ameisenplage") |>
  fetchAll() |>
  slot("collectedMatches") |>
  mutate(year = lubridate::year(pubDate)) |>
  dplyr::select(year) |>
  group_by(year) |>
  summarise(Count = dplyr::n()) |>
  mutate(Freq = mapply(function(f, y) {
    f / corpusStats(kco, paste("pubDate in", y))@tokens
```

```
  }, Count, year)) |>
  dplyr::select(-Count) |>
  complete(year = min(year):max(year), fill = list(Freq = 0)) |>
  plot(type = "l")

## End(Not run)
```

`corpusStats,KorAPConnection-method`

Get corpus size and statistics

Description

Retrieve information about corpus size (documents, tokens, sentences, paragraphs) for the entire corpus or a virtual corpus subset.

Usage

```
## S4 method for signature 'KorAPConnection'
corpusStats(kco, vc = "", verbose = kco@verbose, as.df = FALSE, cacheAs = NULL)
```

Arguments

`kco`	`KorAPConnection()` object (obtained e.g. from `KorAPConnection()`
`vc`	string describing the virtual corpus. An empty string (default) means the whole corpus, as far as it is license-wise accessible.
`verbose`	logical. If `TRUE`, additional diagnostics are printed.
`as.df`	return result as data frame instead of as S4 object?
`cacheAs`	path to an RDS file to keep the result in. If the file exists and records the same call, it is read back instead of contacting the server; otherwise the query is run and its result stored there. Unlike the connection's `cache`, this file belongs to the caller, which is what keeps an analysis reproducible once the corpus has grown or the scores have changed. Defaults to `NULL` (no file).

Value

Object containing corpus statistics with the following information:

`vc` Virtual corpus definition used (empty string for entire corpus)

`documents` Total number of documents in the (virtual) corpus

`tokens` Total number of word tokens in the (virtual) corpus

`sentences` Total number of sentences in the (virtual) corpus

`paragraphs` Total number of paragraphs in the (virtual) corpus

`webUIRequestUrl` URL to view this corpus subset in KorAP web interface

When `as.df=TRUE`, returns a data frame with these columns. When `as.df=FALSE` (default), returns a KorAPCorpusStats object with these values as slots.

Usage

```
# Get statistics for entire corpus
kcon <- KorAPConnection()
stats <- corpusStats(kcon)

# Get statistics for a specific time period
stats <- corpusStats(kcon, "pubDate in 2020")

# Access the number of tokens
stats@tokens
```

Examples

```
## Not run:

kco <- KorAPConnection()

# Get statistics for entire corpus (returns S4 object)
stats <- corpusStats(kco)
stats@tokens  # Access number of tokens

# Get statistics for newspaper texts from 2017 (as data frame)
df <- corpusStats(kco, "pubDate in 2017 & textType=/Zeitung.*/", as.df = TRUE)
df$documents  # Access number of documents

# Compare corpus sizes across years
years <- 2015:2020
sizes <- sapply(years, function(y) {
  corpusStats(kco, paste("pubDate in", y))@tokens
})

## End(Not run)
```

---

dispersion-score-functions

Dispersion measures and user-defined scores

Description

[Experimental]

Usage

```
defaultDispersionFunctions()

## S4 method for signature 'numeric'
dispersionAnalysis(
```

```
  frequencies,
  sizes,
  scoreFunctions = defaultDispersionFunctions()
)

dispersionDP(frequencies, sizes)

dispersionDPnorm(frequencies, sizes)

dispersionJuillandD(frequencies, sizes)

dispersionCarrollD2(frequencies, sizes)

dispersionRosengrenS(frequencies, sizes)

dispersionRange(frequencies, sizes)
```

Arguments

`frequencies`	Numeric vector of observed or estimated hit counts, or a `corpusQuery()` result; see dispersionAnalysis,KorAPQuery-method for the latter.
`sizes`	Numeric vector of corresponding corpus-part sizes.
`scoreFunctions`	Named list of functions with arguments `function(frequencies, sizes)`, each returning one numeric value. As with `defaultAssociationScoreFunctions()`, replace or extend the list to use your own measures. Each function receives the absolute input values, with zero-sized, zero-hit parts removed.

Details

Calculate dispersion from absolute frequencies and corpus-part sizes. The parts must be disjoint; include positive-sized parts without hits. Sizes are token counts (or proportional sizes), not the frequencies of the query. The analysis interface and result structure may change in future releases.

`dispersionDP()` computes `sum(abs(frequencies / sum(frequencies) - sizes / sum(sizes))) / 2`. `dispersionDPnorm()` divides DP by `1 - min(sizes / sum(sizes))`, following the corrected normalization in Lijffijt and Gries (2012). Smaller DP and DPnorm mean more even distribution. Juilland's D, Carroll's D2 and Rosengren's S use the versions for unequal part sizes described by Gries (2020); larger values mean more even distribution. These are independent implementations of the published formulas; Gries' companion script is not included or required. `dispersionRange()` counts parts with hits, not documents unless documents are the parts.

The built-in measures other than range return `NA_real_` if there are no hits or fewer than two positive-sized parts. Range is zero without hits. No measure is assumed to be appropriate for every research question, and the measures are not combined into a rank or adjusted frequency.

Value

`dispersionAnalysis()` returns a one-row tibble, one column per score. `defaultDispersionFunctions()` returns a named list of functions. Individual score functions return a numeric scalar.

Interpretation

Range describes observed reach: the number of parts containing at least one hit. It ignores part sizes and within-part frequencies, and depends on the total number of hits. Broader reach need not mean more even distribution.

DP compares each part's share of hits with its share of corpus tokens. For example, DP = 0.20 means that 20 percent of the hit distribution would have to be reallocated between parts to match the token shares, keeping the total fixed. This is hypothetical hit mass, allowing fractional counts; it is not a percentage of sources or speakers. Smaller is closer, and zero is an exact match. DPnorm rescales DP by its maximum for the given part sizes; the redistribution interpretation applies to DP itself, not directly to DPnorm.

Juilland's D describes variation in relative hit frequencies across parts; Carroll's D2 describes their entropy. Both equal one when each part has the same relative frequency. Rosengren's S compares the hit and token distributions, reaching one when they match. Larger D, D2 and S mean more even distribution under their respective definitions. The measures emphasize different aspects, so they need not rank variants in the same order.

Compare variants within the same corpus and partition. There is no universal cutoff for good dispersion, and these descriptive scores are not significance tests. Averaging ranks would depend on the selected measures, give related measures repeated votes and discard the size of differences. Inspect the underlying part counts and token sizes, and report disagreements explicitly. The leading one or two sources' shares of hits, considered alongside their token shares, can help explain concentration. Concordances are needed to investigate possible house styles, quotation or syndicated material.

References

Gries, Stefan Th. (2008). Dispersions and adjusted frequencies in corpora. International Journal of Corpus Linguistics, 13(4), 403-437. doi:10.1075/ijcl.13.4.02gri.

Gries, Stefan Th. (2020). Analyzing dispersion. In M. Paquot & S. Th. Gries (eds.), A Practical Handbook of Corpus Linguistics, 99-118. doi:10.1007/9783030462161_5. Companion resources: https://www.stgries.info/research/dispersion/links.html.

Lijffijt, Jefrey & Gries, Stefan Th. (2012). Correction to "Dispersions and adjusted frequencies in corpora". International Journal of Corpus Linguistics, 17(1), 147-149. doi:10.1075/ijcl.17.1.08lij.

Juilland, Alphonse G. & Chang-Rodriguez, Eugenio (1964). Frequency dictionary of Spanish words. The Hague: Mouton.

Carroll, John B. (1970). An alternative to Juilland's usage coefficient for lexical frequencies, and a proposal for a standard frequency index (SFI). Computer Studies in the Humanities and Verbal Behavior, 3(2), 61-65.

Rosengren, Inger (1971). The quantitative concept of language and its relation to the structure of frequency dictionaries. Etudes de linguistique appliquee (Nouvelle Serie), 1, 103-127.

Examples

```
dispersionAnalysis(c(1, 2, 3, 4, 5), c(9, 10, 10, 10, 11))
dispersionAnalysis(c(10, 0), c(100, 100), scoreFunctions = list(
  DP = dispersionDP,
  evenness = function(frequencies, sizes) 1 - dispersionDP(frequencies, sizes)
))
```

`dispersionAnalysis,KorAPQuery-method`
Calculate dispersion directly from a corpus query

Description

[Experimental]

Usage

```
## S4 method for signature 'KorAPQuery'
dispersionAnalysis(
  frequencies,
  by = "source",
  regex = NULL,
  scoreFunctions = defaultDispersionFunctions(),
  exactFrequencies = NULL,
  searchHitsSampleLimit = 20000,
  seed = 7,
  verbose = frequencies@korapConnection@verbose,
  cacheAs = NULL
)

## S4 method for signature 'data.frame'
dispersionAnalysis(
  frequencies,
  by = "source",
  regex = NULL,
  scoreFunctions = defaultDispersionFunctions(),
  exactFrequencies = NULL,
  searchHitsSampleLimit = 20000,
  seed = 7,
  verbose = NULL,
  cacheAs = NULL
)
```

Arguments

`frequencies`	A `corpusQuery()` result for one or several queries.
`by`	Metadata field or vector of fields. `"source"` groups DeReKo's annual `corpusTitle` values into sources automatically; `"year"` groups `pubDate` by publication year; `"region"` uses `pubPlaceKey`. Automatic grouping of `pubDate` and `creationDate` uses calendar-year groups.
`regex`	Optional regular expression removing a trailing suffix from metadata values, ending in `$`, e.g. `"[0-9]{4}$"`. For `by = "source"` this is the default. For custom grouping, use a single field such as `by = "corpusTitle"`. With multiple dimensions, use their default grouping.

`scoreFunctions` Named list of replaceable measure functions; see `defaultDispersionFunctions()`.

`exactFrequencies`
By default (`NULL`), reuse an explicit page sample or complete, unjoined metadata containing the requested fields; otherwise request exact counts per group. `TRUE` requests exact counts even after a fetch. `FALSE` permits metadata sampling, controlled by `searchHitsSampleLimit`.

`searchHitsSampleLimit`
Maximum metadata sample size; see `dispersionQuery()`.

`seed` Seed for random page selection.

`verbose` Whether to print progress.

`cacheAs` Optional result file; see `frequencyQuery()`.

Details

Ask a corpus question first, then choose the metadata dimension of its dispersion. The library discovers the groups and their sizes, including groups without query hits. No source lists or classification code are needed. The analysis interface and result structure may change in future releases.

An explicit `fetchSamplePages()` step makes the retrieval scope visible. Recorded pages are reused without drawing again; any missing metadata are retrieved from those same pages. Multiple queries share independently discovered groups and token sizes, seeded by the groups already observed across their fetched metadata. Printing their result shows one column per query variant, labelled by the names of the original query vector.

See the Interpretation section in dispersion-score-functions for what the measures describe, their directions and why they can disagree. Use `distribution` to examine source counts and token sizes alongside the scores.

`by = "year"` treats publication years as discrete corpus parts, using each year's token count as its size and retaining positive-sized years without query hits. The built-in measures ignore temporal order and distances; they describe dispersion across year groups, not a trend over time.

Use a metadata-only query (the default of `corpusQuery()`). A full-text query rewritten to a smaller licensed population is rejected, so that hit counts and corpus-part sizes cannot silently refer to different populations.

All metadata categories are discovered independently of the query hits. The current API does not offer a public distinct-values endpoint, so the library retrieves one representative text's metadata from each remaining group, excluding already known groups until none remain. It does not download the corpus. The resulting partition is cached per server, index revision, virtual corpus, metadata field and grouping rule when connection caching is enabled. Discovery and size queries can take time on the first call, especially for fields with many categories.

`regex` removes a suffix, rather than selecting which categories are included. It must have a corresponding corpus-query regex: use ordinary groups and character classes, without lookarounds. This makes the metadata grouping and the token denominators agree. Advanced arbitrary groupings remain available through `corpusPartition()` and `dispersionQuery()`.

Value

A `KorAPDispersion` tibble. Printing it shows all requested scores. Detailed counts remain in `distribution`; `sampling` records whether the scores are estimated. The complete partitions are also retained as the `partitions` attribute. See `dispersionQuery()` for the other columns.

Examples

```
## Not run:
kco <- KorAPConnection()
q <- corpusQuery(kco, "NutzerInnen", vc = "pubPlaceKey=AT & pubDate in 2022")
dispersionAnalysis(q, by = "source")
# The explicit metadata-field equivalent:
dispersionAnalysis(q, by = "corpusTitle", regex = " [0-9]{4}$")

## End(Not run)
```

---

dispersionQuery,KorAPConnection-method

Query dispersion across corpus metadata

Description

Compute selectable dispersion measures for one or more queries, using exact counts per part or a reproducible metadata sample shared by all partitions.

Usage

```
## S4 method for signature 'KorAPConnection'
dispersionQuery(
  kco,
  query,
  partitions = NULL,
  scoreFunctions = defaultDispersionFunctions(),
  exactFrequencies = TRUE,
  searchHitsSampleLimit = 20000,
  seed = 7,
  ql = "poliqarp",
  verbose = kco@verbose,
  cacheAs = NULL,
  by = "source",
  regex = NULL,
  vc = ""
)
```

Arguments

kco	A KorAPConnection() object.
query	Query string or vector of queries.
partitions	Optional corpusPartition() table, or named list of them. If NULL, discover groups using by, as in dispersionAnalysis,KorAPQuery-method. Explicit partitions must have the same base virtual corpus and KorAP instance.
scoreFunctions	Named list of replaceable measure functions; see defaultDispersionFunctions().

`exactFrequencies`
: If `TRUE` (default), request exact counts for each part, without downloading matches. If `FALSE`, collect match metadata, using at most `searchHitsSampleLimit` hits. No snippets are requested.

`searchHitsSampleLimit`
: Maximum metadata sample size, default 20000. `Inf` requests all match metadata. Samples draw complete pages without replacement, so finite limits are rounded down to multiples of 50 and must be at least 50 unless all hits fit below the limit.

`seed`
: Seed for page selection. The caller's random state is restored.

`ql`
: Query language, passed to `corpusQuery()`.

`verbose`
: Whether to print progress.

`cacheAs`
: Optional result file; see `frequencyQuery()`.

`by`
: Metadata field(s) for automatic grouping, default `"source"`.

`regex`
: Optional trailing-suffix grouping rule for automatic grouping.

`vc`
: Base virtual corpus for automatic grouping. With explicit partitions, their base virtual corpus is used instead.

Details

A sample is a simple random sample of result pages, not independent hits. Every hit has the same page-inclusion probability. The returned distribution retains unobserved parts and reports observed and extrapolated hit counts. Ratios of these estimates provide plug-in estimates of dispersion; nonlinear measures can be strongly biased by within-page clustering, even with many sampled hits. Range from a sample is only a lower bound, not an estimate of the number of occupied parts. No confidence bounds are claimed. Missing or unlisted metadata are counted as excluded hits.

Partition sizes come from independent corpus statistics, never from the sample. Failed, incomplete or timed-out requests abort the analysis rather than becoming zeros. `exactFrequencies = TRUE` never falls back to sampling. Arbitrary `parts` require a membership classifier for metadata sampling. Reuse a partition only with its original base corpus, server and index revision.

Value

A tibble with one row per query and partition, the requested scores, `totalResults` (base-corpus hits), `sampledHits` (metadata rows analyzed), `usedHits`, `excludedHits`, `nParts`, `estimated`, `rangeIsLowerBound`, and list columns `distribution` and `sampling`. Distributions include corpus-part sizes, observed counts, estimated frequencies, relative frequencies and concordance URLs. `sampling` records the method, seed, selected pages, inclusion probability, server and index revision.

Examples

```
## Not run:
kco <- KorAPConnection()
years <- corpusPartition(kco, "year", 2020:2022)
dispersionQuery(kco, "NutzerInnen", years, scoreFunctions = list(
  evenness = function(frequencies, sizes) 1 - dispersionDP(frequencies, sizes)
```

```
))
dispersionQuery(kco, "NutzerInnen", years, exactFrequencies = FALSE,
                searchHitsSampleLimit = 10000, seed = 7)

## End(Not run)
```

`fetchAll,KorAPQuery-method`

Fetch all results of a KorAP query.

Description

`fetchAll` fetches all results of a KorAP query.

Usage

```
## S4 method for signature 'KorAPQuery'
fetchAll(kqo, verbose = kqo@korapConnection@verbose, ...)

## S4 method for signature 'data.frame'
fetchAll(kqo, verbose = NULL, ...)
```

Arguments

`kqo`	object obtained from `corpusQuery()`
`verbose`	print progress information if true
`...`	further arguments passed to `fetchNext()`

Value

The updated `kqo` object with all results in `@collectedMatches`

See Also

Other corpus search functions: `corpusQuery,KorAPConnection-method`, `fetchAnnotations,KorAPQuery-method`, `fetchNext,KorAPQuery-method`

Examples

```
## Not run:
# Fetch all metadata of every query hit for "Ameisenplage" and show a summary
q <- KorAPConnection() |>
  corpusQuery("Ameisenplage") |>
  fetchAll()
q@collectedMatches

# Fetch also all KWICs
q <- KorAPConnection() |> auth() |>
```

```
 corpusQuery("Ameisenplage", metadataOnly = FALSE) |>
 fetchAll()
q@collectedMatches

# Retrieve title and text sigle metadata of all texts published on 1958-03-12
q <- KorAPConnection() |>
 corpusQuery("<base/s=t>", # this matches each text once
    vc = "pubDate in 1958-03-12",
    fields = c("textSigle", "title")
) |>
 fetchAll()
q@collectedMatches

## End(Not run)
```

`fetchAnnotations` *Fetch annotations (generic)*

Description

S4 generic for fetching token annotations for collected matches. See specific methods, e.g. `fetchAnnotations,KorAPQuery-`

Usage

```
fetchAnnotations(kqo, foundry = "tt", overwrite = FALSE,
  verbose = kqo@korapConnection@verbose)
```

Arguments

`kqo`	An object on which an annotation fetching method is defined.
`foundry`	Annotation foundry identifier.
`overwrite`	Logical flag controlling whether to overwrite existing annotations.
`verbose`	Logical flag for progress output.

See Also

`fetchAnnotations,KorAPQuery-method`

`fetchAnnotations,KorAPQuery-method`
Fetch annotations for all collected matches

Description

[Experimental]

Usage

```
## S4 method for signature 'KorAPQuery'
fetchAnnotations(
  kqo,
  foundry = "tt",
  overwrite = FALSE,
  verbose = kqo@korapConnection@verbose
)
```

Arguments

`kqo`	object obtained from `corpusQuery()` with collected matches. Note: the original corpus query should have `metadataOnly = FALSE` for annotation parsing to work.
`foundry`	string specifying the foundry to use for annotations (default: "tt" for TreeTagger)
`overwrite`	logical; if TRUE, re-fetch and replace any existing annotation columns. If FALSE (default), only add missing annotation layers and preserve already fetched ones (e.g., keep POS/lemma from a previous foundry while adding morph from another).
`verbose`	print progress information if true

Details

`fetchAnnotations` fetches annotations (only token annotations, for now) for all matches in the `@collectedMatches` slot of a KorAPQuery object and adds annotation columns directly to the `@collectedMatches` data frame. The method uses the `matchID` from collected matches.

Important: For copyright-restricted corpora, users must be authorized via `auth()` and the initial corpus query must have `metadataOnly = FALSE` to ensure snippets are available for annotation parsing.

The method parses XML snippet annotations and adds linguistic columns to the data frame:

- `pos`: data frame with `left`, `match`, `right` columns, each containing list vectors of part-of-speech tags
- `lemma`: data frame with `left`, `match`, `right` columns, each containing list vectors of lemmas
- `morph`: data frame with `left`, `match`, `right` columns, each containing list vectors of morphological tags

- atokens: data frame with left, match, right columns, each containing list vectors of token text (from annotations)
- annotation_snippet: original XML snippet from the annotation API

Value

The updated kqo object with annotation columns like pos, lemma, morph (and atokens and annotation_snippet) in the @collectedMatches slot. Each column is a data frame with left, match, and right columns containing list vectors of annotations for the left context, matched tokens, and right context, respectively. The original XML snippet for each match is also stored in annotation_snippet.

See Also

Other corpus search functions: corpusQuery,KorAPConnection-method, fetchAll,KorAPQuery-method, fetchNext,KorAPQuery-method

Examples

```
## Not run:

# Fetch annotations for matches using TreeTagger foundry
# Note: Authorization required for copyright-restricted corpora
q <- KorAPConnection() |>
  auth() |>
  corpusQuery("Ameisenplage", metadataOnly = FALSE) |>
  fetchNext(maxFetch = 10) |>
  fetchAnnotations()

# Access linguistic annotations for match i:
pos_tags <- q@collectedMatches$pos
# Data frame with left/match/right columns for POS tags
lemmas <- q@collectedMatches$lemma
# Data frame with left/match/right columns for lemmas
morphology <- q@collectedMatches$morph
# Data frame with left/match/right columns for morphological tags
atokens <- q@collectedMatches$atokens
# Data frame with left/match/right columns for annotation token text
# Original XML snippet for match i
raw_snippet <- q@collectedMatches$annotation_snippet[[i]]

# Access specific components:
# POS tags for the matched tokens in match i
match_pos <- q@collectedMatches$pos$match[[i]]
# Lemmas for the left context in match i
left_lemmas <- q@collectedMatches$lemma$left[[i]]
# Token text for the right context in match i
right_tokens <- q@collectedMatches$atokens$right[[i]]

# Use a different foundry (e.g., MarMoT)
q <- KorAPConnection() |>
  auth() |>
  corpusQuery("Ameisenplage", metadataOnly = FALSE) |>
```

```
  fetchNext(maxFetch = 10) |>
  fetchAnnotations(foundry = "marmot")
q@collectedMatches$pos$left[1] # POS tags for the left context of the first match

## End(Not run)
```

`fetchNext,KorAPQuery-method`
Fetch the next bunch of results of a KorAP query.

Description

`fetchNext` fetches the next bunch of results of a KorAP query.

Usage

```
## S4 method for signature 'KorAPQuery'
fetchNext(
  kqo,
  offset = kqo@nextStartIndex,
  maxFetch = maxResultsPerPage,
  verbose = kqo@korapConnection@verbose,
  randomizePageOrder = FALSE,
  preserveMetadataValues = NULL
)

## S4 method for signature 'data.frame'
fetchNext(kqo, ..., verbose = NULL)
```

Arguments

`kqo`	object obtained from `corpusQuery()`
`offset`	start offset for query results to fetch
`maxFetch`	maximum number of query results to fetch
`verbose`	print progress information if true
`randomizePageOrder`	fetch result pages in pseudo random order if true. Use `set.seed()` to set seed for reproducible results.
`preserveMetadataValues`	Preserve list-valued metadata from the fields API instead of joining multiple values with spaces. Useful when an analysis needs to distinguish multivalued categories. Defaults to the mode recorded by a previous fetch, or `FALSE` for an unfetched query. To change modes, restart retrieval with `fetchAll()`.
`...`	Further fetch arguments applied to each query in a multi-query result.

Value

The kqo input object with updated slots collectedMatches, apiResponse, nextStartIndex, hasMoreMatches With preserveMetadataValues = TRUE, metadata columns with at most one value per match are atomic vectors (missing values become NA). Truly multivalued columns remain lists; a later page may introduce multiple values and promote a previously scalar column to a list.

References

https://ids-pub.bsz-bw.de/frontdoor/index/index/docId/9026

See Also

Other corpus search functions: corpusQuery,KorAPConnection-method, fetchAll,KorAPQuery-method, fetchAnnotations,KorAPQuery-method

Examples

```
## Not run:

q <- KorAPConnection() |>
  corpusQuery("Ameisenplage") |>
  fetchNext()
q@collectedMatches

## End(Not run)
```

---

fetchRest,KorAPQuery-method

Fetches the remaining results of a KorAP query.

Description

Fetches the remaining results of a KorAP query.

Usage

```
## S4 method for signature 'KorAPQuery'
fetchRest(kqo, verbose = kqo@korapConnection@verbose, ...)

## S4 method for signature 'data.frame'
fetchRest(kqo, verbose = NULL, ...)
```

Arguments

kqo	object obtained from corpusQuery()
verbose	print progress information if true
...	further arguments passed to fetchNext()

Value

The updated kqo object with remaining results in `@collectedMatches`

Examples

```
## Not run:

q <- KorAPConnection() |>
  corpusQuery("Ameisenplage") |>
  fetchRest()
q@collectedMatches

## End(Not run)
```

`fetchSamplePages,KorAPQuery-method`

Fetch a reproducible sample of result pages

Description

Select complete result pages without replacement. This is a page (cluster) sample, not a simple random sample of individual hits. Hits within a selected page remain in Lucene index order. A single randomly selected page therefore does not provide independently sampled hits. If all results fit within the limit, all pages are fetched in index order and no sample is drawn.

Usage

```
## S4 method for signature 'KorAPQuery'
fetchSamplePages(
  kqo,
  maxHits = 2000,
  seed = 7,
  verbose = kqo@korapConnection@verbose,
  warn = TRUE
)

## S4 method for signature 'data.frame'
fetchSamplePages(kqo, maxHits = 2000, seed = 7, verbose = NULL, warn = TRUE)
```

Arguments

kqo	A corpusQuery() result, for one or several queries.
maxHits	Maximum hits to retrieve per query. Complete pages are used, so a finite limit is rounded down to a multiple of 50. At least 50 is required unless the complete result fits within maxHits. Inf fetches all results.
seed	Seed for choosing pages; the caller's random state is restored.

verbose	Whether to print progress.
warn	Emit the page-sampling warning, or an informational message if all hits are fetched. Defaults to TRUE; the procedure remains recorded in the result even when this is explicitly disabled.

Details

Unlike taking the first hits, pages are selected from the entire result set. If several pages are available, even one page can be selected at random. Within-page ordering and clustering remain; this distinction matters when interpreting estimates and uncertainty. The actual row count can be below maxHits, particularly when the last, shorter page is selected.

Call fetchAll() to replace a page sample with the complete result. A page sample is not a sequential prefix, so fetchNext() and fetchRest() cannot safely continue it. Call fetchSamplePages() again to draw a new sample.

Value

Updated query object(s), with hits in collectedMatches and page sampling provenance in the korapSampling attribute. dispersionAnalysis() uses this exact sample and its inclusion probability without drawing again. Single-valued metadata columns are atomic vectors, with NA for missing values; genuinely multivalued columns remain lists.

Examples

```
## Not run:
KorAPConnection() |>
  corpusQuery("Gendern") |>
  fetchSamplePages(maxHits = 2000, seed = 7) |>
  dispersionAnalysis(by = "source")

## End(Not run)
```

---

frequencyQuery,KorAPConnection-method

Query frequencies of search expressions in virtual corpora

Description

frequencyQuery combines corpusQuery(), corpusStats() and ci() to compute a tibble with the absolute and relative frequencies and confidence intervals of one or multiple search terms across one or multiple virtual corpora.

Usage

```
## S4 method for signature 'KorAPConnection'
frequencyQuery(
  kco,
  query,
  vc = "",
  conf.level = 0.95,
  as.alternatives = FALSE,
  cacheAs = NULL,
  dispersionBy = NULL,
  dispersionFunctions = defaultDispersionFunctions(),
  dispersionExactFrequencies = TRUE,
  searchHitsSampleLimit = 20000,
  seed = 7,
  ...
)
```

Arguments

kco	KorAPConnection() object (obtained e.g. from KorAPConnection())
query	corpus query string(s) (can be a vector). The query language depends on the ql parameter. Either query must be provided or KorAPUrl.
vc	virtual corpus definition(s) (can be a vector)
conf.level	confidence level of the returned confidence interval (passed through ci() to prop.test()).
as.alternatives	LOGICAL that specifies if the query terms should be treated as alternatives. If as.alternatives is TRUE, the sum over all query hits, instead of the respective vc token sizes is used as total for the calculation of relative frequencies.
cacheAs	path to an RDS file to keep the result in. If the file exists and records the same call, it is read back instead of contacting the server; otherwise the query is run and its result stored there. Unlike the connection's cache, this file belongs to the caller, which is what keeps an analysis reproducible once the corpus has grown or the scores have changed. Defaults to NULL (no file).
dispersionBy	Optional corpusPartition() table or named list of tables. Adds a dispersion list column computed by dispersionQuery(). Each partition's base corpus must match the frequency row's vc.
dispersionFunctions	Named list of replaceable dispersion functions; see defaultDispersionFunctions().
dispersionExactFrequencies	Use exact part counts for the optional dispersion analysis; FALSE uses metadata sampling.
searchHitsSampleLimit	Maximum number of metadata hits to download for the optional dispersion analysis; see dispersionQuery().
seed	Seed for optional dispersion sampling.

... further arguments passed to or from other methods (see corpusQuery()), most notably expand, a logical that decides if query and vc parameters are expanded to all of their combinations. It defaults to TRUE, if query and vc have different lengths, and to FALSE otherwise.

Value

A tibble, with each row containing the following result columns for query and vc combinations:

- **query**: the query string used for the frequency analysis.
- **variant**: query-vector names, when supplied, for display labels.
- **as.alternatives**: denominator mode, retained for plotting helpers.
- **totalResults**: absolute frequency of query matches in the vc.
- **vc**: virtual corpus used for the query.
- **queryDuration**: client-side duration of the request in seconds.
- **webUIRequestUrl**: URL of the corresponding web UI request with respect to query and vc.
- **total**: total number of words in vc.
- **f**: relative frequency of query matches in the vc.
- **conf.low**: lower bound of the confidence interval for the relative frequency, given conf.level.
- **conf.high**: upper bound of the confidence interval for the relative frequency, given conf.level.

Slow queries

Complex queries can take minutes on large corpora, especially on DeLiKo@DNB (https://korap.dnb.de), or exceed the time limit of the server, which leaves the results incomplete. As long as KorAP does not offer sampling by itself, a workaround is to restrict vc to a pseudo-random sample of texts, selected by the last digit of their sigle: "textSigle = /.*5/" selects about a tenth, "textSigle = /.*[0-4]/" about half of the texts. Combine the sample with other conditions using &, e.g. "textType = /.*Krimi.*/ & textSigle = /.*5/". Relative frequencies, and association scores that do not depend on the corpus size, such as logDice, remain estimates for the whole (virtual) corpus, whereas absolute frequencies, and with them log-likelihood scores, shrink with the sample. A sample saves about as much time as it leaves out if the server has recently read the index of the query terms, e.g. for the same query on another virtual corpus, but hardly any time otherwise, when most of the time goes into reading the index from disk. It thus pays off in series of queries on the same terms, such as a frequencyQuery() over many years or a collocationAnalysis(). Repeating a query on the same vc takes no time anyway, as its result comes from the local cache (see the cache parameter of KorAPConnection()). Queries that are fast anyway do not get faster, because selecting the sample takes some time itself.

Examples

```
## Not run:

KorAPConnection(verbose = TRUE) |>
  frequencyQuery(c("Mücke", "Schnake"), paste0("pubDate in ", 2000:2003))

## End(Not run)
```

`geom_freq_bar_ci` *Plot frequency bars with confidence intervals using ggplot2*

Description

[Experimental]

Add a column layer and aligned confidence-interval error bars to a ggplot. Like `geom_freq_by_year_ci()`, this helper uses the values already mapped in the plot: convert a `frequencyQuery()` result with `ipm()` first, or with `percent()` for alternatives. It does not query KorAP or rescale the data.

Usage

```
geom_freq_bar_ci(
  mapping = NULL,
  data = NULL,
  position = "dodge",
  width = 0.9,
  errorbar.width = 0.25,
  errorbar.colour = NULL,
  errorbar.params = list(),
  na.rm = FALSE,
  orientation = "x",
  showCI = TRUE,
  ...
)
```

Arguments

`mapping`	Error-bar bounds, created with `ggplot2::aes()`. Defaults to ymin = conf.low and ymax = conf.high (xmin and xmax for orientation = "y"). Set x, y, fill and group in the main `ggplot2::ggplot()` mapping so both layers inherit them.
`data`	Optional data shared by both layers; otherwise they inherit the plot data.
`position`	"dodge" (default), "identity", or a `ggplot2::position_dodge()` or `ggplot2::position_identity()` object. Both layers use the same position adjustment. An unspecified dodge width is set to the bar width.
`width`	Bar width, default 0.9.
`errorbar.width`	Width of the error-bar caps, default 0.25.
`errorbar.colour`	Optional fixed error-bar colour. The default NULL inherits the plot's colour mapping, or uses a neutral grey if none is mapped. With ggplot2 4, the default grey adapts to the theme's background.
`errorbar.params`	Named list of additional error-bar appearance options, for example list(linewidth = 0.4, alpha = 0.8).
`na.rm`	Remove missing values without a warning, passed to both layers.

orientation	"x" (default) for a categorical x axis and vertical intervals; "y" for a categorical y axis and horizontal intervals. Both layers use the same orientation. An implicit horizontal mapping is rejected.
showCI	Show confidence intervals (default TRUE). FALSE adds only the bar layer and does not require interval bounds.
...	Additional appearance options passed to ggplot2::geom_col(), such as fill, colour, alpha or linewidth.

Details

Map fill or group to a query or corpus variable for grouped comparisons. Map group explicitly when fixed fill or colour settings override a grouping aesthetic, so both layers retain the same groups. Stacked bars are not supported because independently calculated confidence limits cannot be stacked. Missing observations are not converted to zeros; use explicit missing rows to reserve slots for absent query/corpus pairs.

Use orientation = "y" with aes(x = ipm, y = label), or use ggplot2::coord_flip() with the default mapping for horizontal bars. Themes, facets, scales and ggplot2::ggsave() work as usual, including PDF and SVG output.

Value

A list of ggplot2 layers, added to a plot with +.

See Also

hc_freq_bar_ci() for interactive charts; geom_freq_by_year_ci() for time series.

Examples

```
## Not run:
library(ggplot2)
kco <- KorAPConnection("https://korap.dnb.de")
frequencyQuery(kco, "Berlin",
  vc = c(Buchpreis = "award=buchpreis", DeLiKo = "")) |>
  ipm() |>
  ggplot(aes(x = label, y = ipm)) +
  geom_freq_bar_ci() +
  theme_bw()

frequencyQuery(kco, c("Berlin", "Paris"),
  vc = c(Buchpreis = "award=buchpreis", DeLiKo = ""), expand = TRUE) |>
  ipm() |>
  ggplot(aes(x = label, y = ipm, fill = query)) +
  geom_freq_bar_ci() +
  coord_flip()

## End(Not run)
```

`hc_add_onclick_korap_search` *Add KorAP search click events to highchart plots*

Description

[Experimental]

Adds on-click events to data points of highcharts that were constructed with `frequencyQuery()` or `collocationScoreQuery()`. Clicks on data points then launch KorAP web UI queries for the given query term and virtual corpus in a separate tab that is reused for subsequent clicks. Only HTTP(S) links are opened; points without a valid link are not clickable.

Usage

```
hc_add_onclick_korap_search(hc)
```

Arguments

hc	A highchart htmlwidget object generated by e.g. `frequencyQuery()`.

Value

The input highchart object with added on-click events.

See Also

Other highcharter-helpers: `hc_dispersion()`, `hc_freq_bar_ci()`, `hc_freq_by_year_ci()`

Examples

```
## Not run:

library(highcharter)
library(tidyr)

KorAPConnection(verbose = TRUE) %>%
  collocationScoreQuery("Team", "agil", vc = paste("pubDate in", c(2014:2018)),
                        lemmatizeNodeQuery = TRUE, lemmatizeCollocateQuery = TRUE) %>%
                         pivot_longer(c("O", "E")) %>%
  hchart(type="spline", hcaes(label, value, group=name)) %>%
  hc_add_onclick_korap_search()

## End(Not run)
```

`hc_dispersion` *Plot interactive distributions underlying a dispersion analysis*

Description

[Experimental]

Compare each query's share of hits with the corpus-token share of each part. The chart uses stored distributions and makes no requests to KorAP.

Usage

```
hc_dispersion(df, partition = NULL, top = NULL)
```

Arguments

`df`	A query-based `dispersionAnalysis()` or `dispersionQuery()` result.
`partition`	Name of one partition in the result. May be omitted when the result contains only one partition. Variants must use the same parts, token sizes, virtual-corpus definitions, KorAP instance and index revision.
`top`	Number of parts to show individually. NULL (default) shows all calendar years for year/date partitions, and 12 parts otherwise. Parts are ranked by the largest hit share across the variants, with token size breaking ties. Remaining parts are combined into an additional labelled bar group. `Inf` shows all parts. For `year`, `pubDate` and `creationDate` partitions, displayed four-digit years are ordered chronologically. Dispersion scores are not recalculated.

Details

Percentages use hits and tokens within the listed, positive-sized parts, matching the population used by the dispersion measures. Positive-sized parts without hits are retained. A zero-hit variant has no defined hit-share distribution and is identified in the caption without drawing hit-share bars.

Grey bars show corpus-token shares; coloured bars show the query variants. Tooltips distinguish shares of total frequency from relative frequencies per million tokens, and show absolute frequencies and subcorpus sizes. Click individual variant bars to open their stored concordance links. Corpus-token bars and combined remainder bars have no concordance link.

Sample-based results are labelled as estimates, with inclusion probabilities in tooltips. Page clustering can strongly bias these shares. The chart does not provide confidence intervals or an overall ranking of dispersion scores. The interface and display may change in future releases.

Value

A highchart htmlwidget, customizable with highcharter functions.

Display language

Generated chart text and number formatting follow the browser's preferred German or English language, falling back to English. Query names and custom labels are retained. Set `options(rkorap.chart.language = "de")` or `"en"` before creating charts to fix the language for exports; `"auto"` restores browser selection. IPM means instances per million tokens.

See Also

Other highcharter-helpers: `hc_add_onclick_korap_search()`, `hc_freq_bar_ci()`, `hc_freq_by_year_ci()`

Examples

```
## Not run:
d <- KorAPConnection() |>
  corpusQuery(c(star = "[orth='Nutzer*innen']",
                colon = "[orth='Nutzer:innen']")) |>
  fetchSamplePages(maxHits = Inf) |>
  dispersionAnalysis(by = "source")
hc_dispersion(d, top = 12)

## End(Not run)
```

---

hc_freq_bar_ci *Plot interactive frequency bars with confidence intervals*

Description

[Experimental]

Compare relative frequencies from a `frequencyQuery()` result. Frequencies and confidence limits are converted to instances per million tokens, or to percentages for alternatives. The function uses the stored results without making further requests to KorAP.

Usage

```
hc_freq_bar_ci(
  df,
  as.alternatives = NULL,
  ylabel = if (as.alternatives) "%" else "IPM",
  horizontal = TRUE,
  log = FALSE,
  showCI = TRUE,
  ...
)
```

Arguments

df	Unconverted data frame returned by `frequencyQuery()`. Do not call `ipm()` or `percent()` first.
as.alternatives	Denominator mode. NULL (default) inherits the mode recorded by `frequencyQuery()`. An explicit conflicting mode is rejected. For older or manually constructed tables without a recorded mode, the default is FALSE. Denominators are never recalculated from displayed rows.
ylabel	Value-axis label, default "IPM" (instances per million tokens) or "%" for alternatives.
horizontal	Draw horizontal bars (default). FALSE draws columns.
log	Use a logarithmic value axis. Non-positive frequencies and intervals touching zero are omitted, with a note in the caption. Use this argument rather than changing only the axis type with hc_yAxis().
showCI	Show confidence intervals (default TRUE). FALSE omits error bars and confidence-interval tooltip lines; interval columns are not required.
...	Additional bar-series options passed to `highcharter::hc_add_series()`, such as dataLabels, color or borderWidth. The data, type, id and linkedTo options are managed by this helper.

Details

A single query is compared across virtual corpora; multiple queries in one corpus are compared across queries. With multiple queries and corpora, each corpus is a category with one bar per query. Categories and query groups keep their first-appearance order. A label column, supplied by named virtual corpora, is used for corpus labels.

Each query/corpus pair must occur at most once. Missing combinations and rows without a frequency or a positive denominator appear as gaps, not as zero counts. Zero hits in a non-empty corpus remain visible with their confidence interval. Error bars use the confidence limits supplied in df, including the confidence level chosen in frequencyQuery().

Tooltips show absolute and relative frequencies, corpus size (or the total frequency of alternatives) and confidence intervals. Clicking a bar opens its stored KorAP concordance link, when available.

Value

A highchart htmlwidget, customizable with highcharter functions.

Display language

Generated chart text and number formatting follow the browser's preferred German or English language, falling back to English. Query names and custom labels are retained. Set `options(rkorap.chart.language = "de")` or `"en"` before creating charts to fix the language for exports; `"auto"` restores browser selection. IPM means instances per million tokens.

See Also

geom_freq_bar_ci() for ggplot2 figures suitable for print.

Other highcharter-helpers: hc_add_onclick_korap_search(), hc_dispersion(), hc_freq_by_year_ci()

Examples

```
## Not run:
kco <- KorAPConnection("https://korap.dnb.de")
frequencyQuery(kco, "Berlin",
  vc = c(Buchpreis = "award=buchpreis", DeLiKo = "")) |>
  hc_freq_bar_ci()

frequencyQuery(kco, c("Iowa", "Berlin"), vc = "award=buchpreis") |>
  hc_freq_bar_ci(horizontal = FALSE)

frequencyQuery(kco, c("Berlin", "Paris"),
  vc = c(Buchpreis = "award=buchpreis", DeLiKo = ""), expand = TRUE) |>
  hc_freq_bar_ci()

## End(Not run)
```

---

hc_freq_by_year_ci *Plot interactive frequency curves with confidence intervals*

Description

Convenience function for plotting typical frequency by year graphs with confidence intervals using highcharter.

Warning: This function may be moved to a new package.

Usage

```
hc_freq_by_year_ci(
  df,
  as.alternatives = NULL,
  ylabel = if (as.alternatives) "%" else "IPM",
  smooth = FALSE,
  showCI = TRUE,
  ...
)
```

Arguments

df	data frame like the value of a frequencyQuery()
as.alternatives	NULL (default) inherits the denominator mode from frequencyQuery(). An explicit conflicting mode is rejected. Older tables without a recorded mode default to FALSE.

ylabel	defaults to % if as.alternatives is TRUE and to IPM (instances per million tokens) otherwise.
smooth	Draw curved connections using Highcharts' spline and areasplinerange types (default FALSE). This only changes the display; observations remain at the measured years. The interpolated band does not supply confidence intervals for dates between observations or a trend test.
showCI	Show confidence bands (default TRUE). FALSE omits bands and confidence-interval tooltip lines; interval columns are not required.
...	additional arguments passed to highcharter::hc_add_series()

Value

A highchart htmlwidget object containing the frequency plot.

Display language

Generated chart text and number formatting follow the browser's preferred German or English language, falling back to English. Query names and custom labels are retained. Set options(rkorap.chart.language = "de") or "en" before creating charts to fix the language for exports; "auto" restores browser selection. IPM means instances per million tokens.

See Also

Other highcharter-helpers: hc_add_onclick_korap_search(), hc_dispersion(), hc_freq_bar_ci()

Examples

```
## Not run:

year <- c(1990:2018)
alternatives <- c("macht []{0,3} Sinn", "ergibt []{0,3} Sinn")
KorAPConnection(verbose = TRUE) %>%
  frequencyQuery(query = alternatives,
                 vc = paste("textType = /Zeit.*/ & pubDate in", year),
                 as.alternatives = TRUE) %>%
  hc_freq_by_year_ci(as.alternatives = TRUE)


kco <- KorAPConnection(verbose = TRUE)
expand_grid(
  condition = c("idsColumn = /Wirtschaft.*/", "idsColumn != /Wirtschaft.*/"),
  year = (2005:2011)
) %>%
  cbind(frequencyQuery(
    kco,
    "[tt/l=Heuschrecke]",
    paste0(.$condition, " & pubDate in ", .$year)
  ))  %>%
  hc_freq_by_year_ci()
```

```
## End(Not run)
```

KorAPConnection-class *Connect to KorAP Server*

Description

KorAPConnection() creates a connection to a KorAP server for corpus queries. This is your starting point for all corpus analysis tasks.

Usage

```
KorAPConnection(
  KorAPUrl = defaultKorAPUrl(),
  apiVersion = "v1.0",
  apiUrl,
  accessToken = getAccessToken(KorAPUrl),
  oauthClient = NULL,
  oauthScope = "search match_info",
  authorizationSupported = TRUE,
  userAgent = "R-KorAP-Client",
  timeout = 240,
  verbose = FALSE,
  cache = TRUE
)
```

Arguments

KorAPUrl	URL of the web user interface of the KorAP server instance you want to access. Defaults to the environment variable KORAP_URL if set and to the IDS Mannheim KorAP main instance to query DeReKo, otherwise. In order to access the KorAP instance at the German National Library (DNB) to query the contemporary fiction corpus DeLiKo@DNB, for example, set KorAPUrl to https://korap.dnb.de/.
apiVersion	which version of KorAP's API you want to connect to. Defaults to "v1.0".
apiUrl	URL of the KorAP web service. If not provided, it will be constructed from KorAPUrl and apiVersion.
accessToken	OAuth2 access token. For queries on corpus parts with restricted access (e.g. textual queries on IPR protected data), you need to authorize your application with an access token. You can obtain an access token in the OAuth settings of your KorAP web interface. More details are explained in the authorization section of the RKorAPClient Readme on GitHub. To use authorization based on an access token in subsequent queries, initialize your KorAP connection with:

```
kco <- KorAPConnection(accessToken="<access token>")
```

In order to make the API token persistent for the currently used `KorAPUrl` (you can have one token per KorAPUrl / KorAP server instance), use:

```
persistAccessToken(kco)
```

This will store it in your keyring using the keyring::keyring-package. Subsequent KorAPConnection() calls will then automatically retrieve the token from your keying. To stop using a persisted token, call `clearAccessToken(kco)`. Please note that for DeReKo, authorized queries will behave differently inside and outside the IDS, because of the special license situation. This concerns also cached results which do not take into account from where a request was issued. If you experience problems or unexpected results, please try `kco <- KorAPConnection(cache=FALSE)` or use `clearCache()` to clear the cache completely.

An alternative to using an access token is to use a browser-based oauth2 workflow to obtain an access token. This can be done with the `auth()` method.

`oauthClient` OAuth2 client object.

`oauthScope` OAuth2 scope. Defaults to "search match_info".

`authorizationSupported`
logical that indicates if authorization is supported/necessary for the current KorAP instance. Automatically set during initialization.

`userAgent` user agent string. Defaults to "R-KorAP-Client".

`timeout` timeout in seconds for API requests (this does not influence server internal timeouts). Defaults to 240 seconds.

`verbose` logical that decides whether following operations will default to be verbose. Defaults to FALSE. If not explicitly provided, this can be overridden via environment variable `KORAP_VERBOSE` (accepted true-ish values: 1, true, yes, on) or R option `rkorap.verbose` (logical).

`cache` logical that decides if API calls are cached locally. You can clear the cache with `clearCache()`. Defaults to TRUE.

Details

Use `KorAPConnection()` to connect, then `corpusQuery()` to search, and `fetchAll()` to retrieve results. For authorized access to restricted corpora, use `auth()` or provide an `accessToken`.

The KorAPConnection object contains various configuration slots for advanced users: KorAPUrl (server URL), apiVersion, accessToken (OAuth2 token), timeout (request timeout), verbose (logging), cache (local caching), and other technical parameters. Most users can ignore these implementation details.

Value

`KorAPConnection()` object that can be used e.g. with `corpusQuery()`

Basic Workflow

```
# Connect to KorAP
kcon <- KorAPConnection()

# Search for a term
query <- corpusQuery(kcon, "Ameisenplage")

# Get all results
results <- fetchAll(query)
```

Authorization

For access to restricted corpora, authorize your connection:

```
kcon <- KorAPConnection() |> auth()
```

See Also

Other initialization functions: `auth,KorAPConnection-method`, `clearAccessToken,KorAPConnection-method`, `persistAccessToken,KorAPConnection-method`

---

mergeDuplicateCollocates

Merge duplicate collocate rows and re-calculate association scores and URLs. Useful if collocation analyses were performed separately for collocates on the left and right side of a node.

Description

Merge duplicate collocate rows and re-calculate association scores and URLs. Useful if collocation analyses were performed separately for collocates on the left and right side of a node.

Usage

```
mergeDuplicateCollocates(..., smoothingConstant = 0.5)
```

Arguments

`...`	tibbles with collocate rows returned from `collocationAnalysis()`
`smoothingConstant`	original smoothing constant (to be added only once to the observed values)

Value

tibble with unique collocate rows

persistAccessToken,KorAPConnection-method
Persist current access token in keyring

Description

Persist current access token in keyring

Usage

```
## S4 method for signature 'KorAPConnection'
persistAccessToken(kco, accessToken = kco@accessToken)
```

Arguments

kco	KorAPConnection object
accessToken	access token to be persisted. If not supplied, the current access token of the KorAPConnection object will be used.

Value

KorAPConnection object.

See Also

clearAccessToken(), auth()

Other initialization functions: KorAPConnection-class, auth,KorAPConnection-method, clearAccessToken,KorAPConn

Examples

```
## Not run:
kco <- KorAPConnection(accessToken = "e739u6eOzkwADQPdVChxFg")
persistAccessToken(kco)

kco <- KorAPConnection() %>%
  auth(app_id = "<my application id>") %>%
  persistAccessToken()

## End(Not run)
```

print.KorAPDispersion *Print dispersion measures without hiding them behind metadata columns*

Description

Print dispersion measures without hiding them behind metadata columns

Usage

```
## S3 method for class 'KorAPDispersion'
print(x, ..., width = Inf, n = Inf)
```

Arguments

x	A dispersion result.
...	Passed to the tibble print method.
width	Output width. Defaults to Inf so score columns stay visible.
n	Number of rows to print. Defaults to Inf to show all measures.

Value

The input, invisibly.

synsemanticStopwords *Preliminary synsemantic stopwords function*

Description

[Experimental]

Preliminary synsemantic stopwords function to be used in collocation analysis.

Usage

```
synsemanticStopwords(...)
```

Arguments

...	future arguments for language detection

Details

Currently only suitable for German. See stopwords package for other languages.

Value

Vector of synsemantic stopwords.

See Also

Other collocation analysis functions: association-score-functions, collocationAnalysis,KorAPConnection-method
collocationScoreQuery,KorAPConnection-method

`textMetadata,KorAPConnection-method`
Retrieve metadata for a text, identified by its sigle (id)

Description

Retrieves metadata for a text, identified by its sigle (id) using the corresponding KorAP API (see Kustvakt Wiki). To retrieve the metadata for every text in a virtual corpus, use `corpusQuery()` with `<base/s=t>` as query, instead.

Usage

```
## S4 method for signature 'KorAPConnection'
textMetadata(kco, textSigle, verbose = kco@verbose, cacheAs = NULL)
```

Arguments

kco	`KorAPConnection()` object (obtained e.g. from `KorAPConnection()`)
textSigle	unique text id (concatenation of corpus, document and text ids, separated by `/`, e.g.) or vector thereof
verbose	logical. If `TRUE`, additional diagnostics are printed. Defaults to `kco@verbose`.
cacheAs	path to an RDS file to keep the result in. If the file exists and records the same call, it is read back instead of contacting the server; otherwise the query is run and its result stored there. Unlike the connection's `cache`, this file belongs to the caller, which is what keeps an analysis reproducible once the corpus has grown or the scores have changed. Defaults to `NULL` (no file).

Value

Tibble with columns for each metadata property. In case of errors, such as non-existing texts/sigles, the tibble will also contain a column called `errors`. If there are metadata columns you cannot make sense of, please ignore them. The function simply returns all the metadata it gets from the server.

Examples

```
## Not run:
KorAPConnection() |> textMetadata(c("WUD17/A97/08542", "WUD17/B96/57558", "WUD17/A97/08541"))

## End(Not run)
```

`withCachedResults` *Take cached results as they are for the duration of an expression*

Description

Sets the cacheAs mode (see cacheAs) while `expr` is evaluated and puts it back afterwards, so that a whole document can be knitted from the files that are there, without a stray `options()` call outliving it.

Usage

```
withCachedResults(expr, mode = c("reuse", "offline"))
```

Arguments

`expr`	the code to evaluate
`mode`	`"reuse"` to take what the files hold, `"offline"` to refuse computing anything that is not in one already

Value

the value of `expr`

See Also

Other cacheAs: `blessCacheAs()`, `cacheAsInfo()`

Examples

```
## Not run:
withCachedResults({
  ca <- KorAPConnection() |> collocationAnalysis("Klima", cacheAs = "klima.rds")
  freq <- KorAPConnection() |> frequencyQuery("Klima", cacheAs = "klima-freq.rds")
})

## End(Not run)
```

Index

∗ **Annotations**
fetchAnnotations,KorAPQuery-method, 35
∗ **association-score-functions**
association-score-functions, 3
∗ **cacheAs**
blessCacheAs, 6
cacheAsInfo, 7
withCachedResults, 57
∗ **collocation analysis functions**
association-score-functions, 3
collocationAnalysis,KorAPConnection-method, 12
collocationScoreQuery,KorAPConnection-method, 17
synsemanticStopwords, 55
∗ **connection-initialization**
clearCache,KorAPConnection-method, 11
∗ **corpus analysis**
corpusStats,KorAPConnection-method, 25
∗ **corpus search functions**
corpusQuery,KorAPConnection-method, 21
fetchAll,KorAPQuery-method, 33
fetchAnnotations,KorAPQuery-method, 35
fetchNext,KorAPQuery-method, 37
∗ **frequency analysis**
frequencyQuery,KorAPConnection-method, 40
∗ **highcharter-helpers**
hc_add_onclick_korap_search, 45
hc_dispersion, 46
hc_freq_bar_ci, 47
hc_freq_by_year_ci, 49
∗ **initialization functions**
auth,KorAPConnection-method, 5
clearAccessToken,KorAPConnection-method, 10
KorAPConnection-class, 51
persistAccessToken,KorAPConnection-method, 54

association-score-functions, 3
auth *(*auth,KorAPConnection-method*)*, 5
auth(), *35*, *52*, *54*
auth,KorAPConnection-method, 5

blessCacheAs, 6
blessCacheAs(), *7*, *57*

cacheAs, *6, 7*, *57*
cacheAsInfo, 7
cacheAsInfo(), *6, 7*, *57*
ci, 8
ci(), *16*, *40, 41*
clearAccessToken *(*clearAccessToken,KorAPConnection-method*)*, 10
clearAccessToken(), *6*, *54*
clearAccessToken,KorAPConnection-method, 10
clearCache *(*clearCache,KorAPConnection-method*)*, 11
clearCache(), *52*
clearCache,KorAPConnection-method, 11
collocationAnalysis *(*collocationAnalysis,KorAPConnection-method*)*, 12
collocationAnalysis(), *4*, *17*, *23*, *42*, *53*
collocationAnalysis,KorAPConnection-method, 12
collocationScoreQuery *(*collocationScoreQuery,KorAPConnection-method*)*, 17
collocationScoreQuery(), *3, 4*, *14*, *45*

collocationScoreQuery,KorAPConnection-method, 17
corpusPartition, 20
corpusPartition(), *30, 31*, *41*
corpusQuery (corpusQuery,KorAPConnection-method), 21
corpusQuery(), *27*, *29, 30*, *32, 33*, *35*, *37–40*, *42*, *52*, *56*
corpusQuery,KorAPConnection-method, 21
corpusStats (corpusStats,KorAPConnection-method), 25
corpusStats(), *24*, *40*
corpusStats,KorAPConnection-method, 25

defaultAssociationScoreFunctions (association-score-functions), 3
defaultAssociationScoreFunctions(), *27*
defaultDispersionFunctions (dispersion-score-functions), 26
defaultDispersionFunctions(), *30, 31*, *41*
dispersion-score-functions, 26, *30*
dispersionAnalysis (dispersion-score-functions), 26
dispersionAnalysis(), *23*, *40*, *46*
dispersionAnalysis,data.frame-method (dispersionAnalysis,KorAPQuery-method), 29
dispersionAnalysis,KorAPQuery-method, *21*, *27*, 29, *31*
dispersionAnalysis,numeric-method (dispersion-score-functions), 26
dispersionCarrollD2 (dispersion-score-functions), 26
dispersionDP (dispersion-score-functions), 26
dispersionDPnorm (dispersion-score-functions), 26
dispersionJuillandD (dispersion-score-functions), 26
dispersionQuery (dispersionQuery,KorAPConnection-method), 31
dispersionQuery(), *30*, *41*, *46*
dispersionQuery,KorAPConnection-method, 31
dispersionRange (dispersion-score-functions), 26
dispersionRosengrenS (dispersion-score-functions), 26

fetchAll (fetchAll,KorAPQuery-method), 33
fetchAll(), *23, 24*, *37*, *40*
fetchAll,data.frame-method (fetchAll,KorAPQuery-method), 33
fetchAll,KorAPQuery-method, 33
fetchAnnotations, 34
fetchAnnotations,KorAPQuery-method, 35
fetchNext (fetchNext,KorAPQuery-method), 37
fetchNext(), *23, 24*, *33*, *38*, *40*
fetchNext,data.frame-method (fetchNext,KorAPQuery-method), 37
fetchNext,KorAPQuery-method, 37
fetchRest (fetchRest,KorAPQuery-method), 38
fetchRest(), *24*, *40*
fetchRest,data.frame-method (fetchRest,KorAPQuery-method), 38
fetchRest,KorAPQuery-method, 38
fetchSamplePages (fetchSamplePages,KorAPQuery-method), 39
fetchSamplePages(), *23*, *30*
fetchSamplePages,data.frame-method (fetchSamplePages,KorAPQuery-method), 39
fetchSamplePages,KorAPQuery-method, 39
frequencyQuery (frequencyQuery,KorAPConnection-method), 40

frequencyQuery(), 9, 17, 20, 23, 30, 32, 42, 43, 45, 47–49
frequencyQuery,KorAPConnection-method, 40

geom_freq_bar_ci, 43
geom_freq_bar_ci(), 49
geom_freq_by_year_ci (ci), 8
geom_freq_by_year_ci(), 43, 44
ggplot2::aes(), 43
ggplot2::coord_flip(), 44
ggplot2::geom_col(), 44
ggplot2::ggplot(), 43
ggplot2::ggsave(), 44
ggplot2::position_dodge(), 43
ggplot2::position_identity(), 43

hc_add_onclick_korap_search, 45
hc_add_onclick_korap_search(), 47, 49, 50
hc_dispersion, 46
hc_dispersion(), 45, 49, 50
hc_freq_bar_ci, 47
hc_freq_bar_ci(), 44, 45, 47, 50
hc_freq_by_year_ci, 49
hc_freq_by_year_ci(), 45, 47, 49
highcharter::hc_add_series(), 48, 50

ipm (ci), 8
ipm(), 43, 48

keyring::keyring-package, 52
KorAPConnection (KorAPConnection-class), 51
KorAPConnection(), 13, 17, 18, 20, 22–25, 31, 41, 42, 52, 56
KorAPConnection-class, 51
KorAPQuery(), 23

ll (association-score-functions), 3
logDice (association-score-functions), 3

make.names(), 16
mergeDuplicateCollocates, 53
mi2 (association-score-functions), 3
mi3 (association-score-functions), 3
misc-functions (ci), 8

percent (ci), 8
percent(), 43, 48
persistAccessToken (persistAccessToken,KorAPConnection-method), 54
persistAccessToken(), 6, 11
persistAccessToken,KorAPConnection-method, 54
pmi, 18
pmi (association-score-functions), 3
print.KorAPDispersion, 55
prop.test(), 8, 41

queryStringToLabel (ci), 8

set.seed(), 37
synsemanticStopwords, 55
synsemanticStopwords(), 5, 17, 19

textMetadata (textMetadata,KorAPConnection-method), 56
textMetadata,KorAPConnection-method, 56

withCachedResults, 57
withCachedResults(), 7