Package 'receptiviti'

May 5, 2023

Type Package

Title Text Analysis Through the 'Receptiviti' API

Version 0.1.4

Description Send text to the <https://www.receptiviti.com> API to be scored by all available frameworks.

License MIT + file LICENSE

Imports curl, jsonlite, digest, arrow, dplyr, future.apply, progressr

Suggests testthat (>= 3.0.0), knitr, rmarkdown, future

Config/testthat/edition 3

RoxygenNote 7.2.3

URL https://receptiviti.github.io/receptiviti-r/, https://github.com/Receptiviti/receptiviti-r

BugReports https://github.com/Receptiviti/receptiviti-r/issues

NeedsCompilation no

Author Receptiviti Inc. [fnd, cph], Kent English [cre], Micah Iserman [aut, ctr]

Maintainer Kent English <kenglish@receptiviti.com>

Repository CRAN

Date/Publication 2023-05-05 20:00:02 UTC

R topics documented:

`receptiviti` *Receptiviti API*

Description

The main function to access the Receptiviti API.

Usage

```
receptiviti(text, output = NULL, id = NULL, text_column = NULL,
  id_column = NULL, file_type = "txt", return_text = FALSE,
  api_args = getOption("receptiviti.api_args", list()),
  frameworks = getOption("receptiviti.frameworks", "all"),
  framework_prefix = TRUE, as_list = FALSE, bundle_size = 1000,
  bundle_byte_limit = 7500000, collapse_lines = FALSE, retry_limit = 50,
  clear_cache = FALSE, clear_scratch_cache = TRUE, request_cache = TRUE,
  cores = detectCores() - 1, use_future = FALSE, in_memory = TRUE,
  verbose = FALSE, overwrite = FALSE, compress = FALSE,
  make_request = TRUE, text_as_paths = FALSE,
  cache = Sys.getenv("RECEPTIVITI_CACHE"), cache_overwrite = FALSE,
  cache_format = Sys.getenv("RECEPTIVITI_CACHE_FORMAT", "parquet"),
  key = Sys.getenv("RECEPTIVITI_KEY"),
  secret = Sys.getenv("RECEPTIVITI_SECRET"),
  url = Sys.getenv("RECEPTIVITI_URL"))

receptiviti_status(url = Sys.getenv("RECEPTIVITI_URL"),
  key = Sys.getenv("RECEPTIVITI_KEY"),
  secret = Sys.getenv("RECEPTIVITI_SECRET"), verbose = TRUE,
  include_headers = FALSE)
```

Arguments

`text`	A character vector with text to be processed, path to a directory containing files, or a vector of file paths. If a single path to a directory, each file is collapsed to a single text. If a path to a file or files, each line or row is treated as a separate text, unless `collapse_lines` is `TRUE`.
`output`	Path to a `.csv` file to write results to. If this already exists, set `overwrite` to `TRUE` to overwrite it.
`id`	Vector of unique IDs the same length as `text`, to be included in the results.
`text_column, id_column`	Column name of text/id, if `text` is a matrix-like object, or a path to a csv file.
`file_type`	File extension to search for, if `text` is the path to a directory containing files to be read in.
`return_text`	Logical; if `TRUE`, `text` is included as the first column of the result.
`api_args`	A list of additional arguments to pass to the API (e.g., `list(sallee_mode = "sparse")`). Defaults to the `receptiviti.api_args` option.

frameworks — A vector of frameworks to include results from. Texts are always scored with all available framework – this just specifies what to return. Defaults to `all`, to return all scored frameworks. Can be set by the `receptiviti.frameworks` option (e.g., `options(receptiviti.frameworks = c("liwc", "sallee"))`).

framework_prefix — Logical; if `FALSE`, will remove the framework prefix from column names, which may result in duplicates. If this is not specified, and 1 framework is selected, or `as_list` is `TRUE`, will default to remove prefixes.

as_list — Logical; if `TRUE`, returns a list with frameworks in separate entries.

bundle_size — Number of texts to include in each request; between 1 and 1,000.

bundle_byte_limit — Memory limit (in bytes) of each bundle, under `1e7` (10 MB, which is the API's limit). May need to be lower than the API's limit, depending on the system's requesting library.

collapse_lines — Logical; if `TRUE`, and `text` contains paths to files, each file is treated as a single text.

retry_limit — Maximum number of times each request can be retried after hitting a rate limit.

clear_cache — Logical; if `TRUE`, will clear any existing files in the cache. Use `cache_overwrite` if you want fresh results without clearing or disabling the cache. Use `cache = FALSE` to disable the cache.

clear_scratch_cache — Logical; if `FALSE`, will preserve the bundles written when `in_memory` is `TRUE`, after the request has been made.

request_cache — Logical; if `FALSE`, will always make a fresh request, rather than using the response from a previous identical request.

cores — Number of CPU cores to split bundles across, if there are multiple bundles. See the Parallelization section.

use_future — Logical; if `TRUE`, uses a `future` back-end to process bundles, in which case, parallelization can be controlled with the `plan` function (e.g., `plan("multisession")` to use multiple cores); this is required to see progress bars when using multiple cores. See the Parallelization section.

in_memory — Logical; if `FALSE`, will write bundles to temporary files, and only load them as they are being requested.

verbose — Logical; if `TRUE`, will show status messages.

overwrite — Logical; if `TRUE`, will overwrite an existing `output` file.

compress — Logical; if `TRUE`, will save as an `xz`-compressed file.

make_request — Logical; if `FALSE`, a request is not made. This could be useful if you want to be sure and load from one of the caches, but aren't sure that all results exist there; it will error out if it encounters texts it has no other source for.

text_as_paths — Logical; if `TRUE`, ensures `text` is treated as a vector of file paths. Otherwise, this will be determined if there are no `NA`s in `text` and every entry is under 500 characters long.

cache — Path to a directory in which to save unique results for reuse; defaults to `Sys.getenv("RECEPTIVITI_CACH` See the Cache section for details.

cache_overwrite
: Logical; if TRUE, will write results to the cache without reading from it. This could be used if you want fresh results to be cached without clearing the cache.

cache_format
: Format of the cache database; see FileFormat. Defaults to Sys.getenv("RECEPTIVITI_CACHE_FORMAT"

key
: API Key; defaults to Sys.getenv("RECEPTIVITI_KEY").

secret
: API Secret; defaults to Sys.getenv("RECEPTIVITI_SECRET").

url
: API endpoint; defaults to Sys.getenv("RECEPTIVITI_URL"), which defaults to "https://api.receptiviti.com/".

include_headers
: Logical; if TRUE, receptiviti_status's verbose message will include the HTTP headers.

Value

A data.frame with columns for text (if return_text is TRUE; the originally entered text), id (if one was provided), text_hash (the MD5 hash of the text), a column each for relevant entries in api_args, and scores from each included framework (e.g., summary.word_count and liwc.i). If as_list is TRUE, returns a list with a named entry containing such a data.frame for each framework.

Cache

By default, results for unique texts are saved in an Arrow database in the cache location (Sys.getenv("RECEPTIVITI_CACHE" and are retrieved with subsequent requests. This ensures that the exact same texts are not re-sent to the API. This does, however, add some processing time and disc space usage.

If a cache location is not specified, a default directory (receptiviti_cache) will be looked for in the system's temporary directory (which is usually the parent of tempdir()). If this does not exist, you will be asked if it should be created. You can disable this prompt with the receptiviti.cache_prompt option (options(receptiviti.cache_prompt = FALSE)).

The cache_format arguments (or the RECEPTIVITI_CACHE_FORMAT environment variable) can be used to adjust the format of the cache.

You can use the cache independently with open_database(Sys.getenv("RECEPTIVITI_CACHE")).

You can set the cache argument to FALSE to prevent the cache from being used, which might make sense if you don't expect to need to reprocess it.

You can also set the clear_cache argument to TRUE to clear the cache before it is used again, which may be useful if the cache has gotten big, or you know new results will be returned. Even if a cached result exists, it will be reprocessed if it does not have all of the variables of new results, but this depends on there being at least 1 uncached result. If, for instance, you add a framework to your account and want to reprocess a previously processed set of texts, you would need to first clear the cache.

Either way, duplicated texts within the same call will only be sent once.

The request_cache argument controls a more temporary cache of each bundle request. This is cleared when the R session ends. You might want to set this to FALSE if a new framework becomes available on your account and you want to process a set of text you already processed in the current R session without restarting.

Another temporary cache is made when `in_memory` is `FALSE`, which is the default when processing in parallel (when `cores` is over `1` or `use_future` is `TRUE`). This contains a file for each unique bundle, which is read by as needed by the parallel workers.

Parallelization

`texts` are split into bundles based on the `bundle_size` argument. Each bundle represents a single request to the API, which is why they are limited to 1000 texts and a total size of 10 MB. When there is more than one bundle and either `cores` is greater than 1 or `use_future` is `TRUE` (and you've externally specified a `plan`), bundles are processed by multiple cores.

Using `future` also allows for progress bars to be specified externally with `handlers`; see examples.

Examples

```
## Not run:

# check that the API is available, and your credentials work
receptiviti_status()

# score a single text
single <- receptiviti("a text to score")

# score multiple texts, and write results to a file
multi <- receptiviti(c("first text to score", "second text"), "filename.csv")

# score many texts in separate files
## defaults to look for .txt files
file_results <- receptiviti("./path/to/txt_folder")

## could be .csv
file_results <- receptiviti(
  "./path/to/csv_folder",
  text_column = "text", file_type = "csv"
)

# score many texts from a file, with a progress bar
## set up cores and progress bar (only necessary if you want the progress bar)
future::plan("multisession")
progressr::handlers(global = TRUE)
progressr::handlers("progress")

## make request
results <- receptiviti(
  "./path/to/largefile.csv",
  text_column = "text", use_future = TRUE
)

## End(Not run)
```

Index