Package 'mscstexta4r'

October 13, 2022

Type Package

Title R Client for the Microsoft Cognitive Services Text Analytics REST API

Version 0.1.2

Maintainer Phil Ferriere <pferriere@hotmail.com>

Description R Client for the Microsoft Cognitive Services Text Analytics REST API, including Sentiment Analysis, Topic Detection, Language Detection, and Key Phrase Extraction. An account MUST be registered at the Microsoft Cognitive Services website <https://www.microsoft.com/cognitive-services/> in order to obtain a (free) API key. Without an API key, this package will not work properly.

License MIT + file LICENSE

URL https://github.com/philferriere/mscstexta4r

BugReports http://www.github.com/philferriere/mscstexta4r/issues

VignetteBuilder knitr

Imports methods, httr, jsonlite, pander, stringi, dplyr, utils

Suggests knitr, rmarkdown, testthat, mscsweblm4r

SystemRequirements A valid account MUST be registered with Microsoft's Cognitive Services website <https://www.microsoft.com/cognitive-services/> in order to obtain a (free) API key. Without an API key, this package will not work properly.

NeedsCompilation no

RoxygenNote 5.0.1

Author Phil Ferriere [aut, cre]

Repository CRAN

Date/Publication 2016-06-23 00:52:35

R topics documented:

`mscstexta4r` *R Client for the Microsoft Cognitive Services Text Analytics REST API*

Description

mscstexta4r is a client/wrapper/interface for the Microsoft Cognitive Services (MSCS) Text Analytics (Text Analytics) REST API. To use this package, you MUST have a valid account with https://www.microsoft.com/cognitive-services. Once you have an account, Microsoft will provide you with a (free) API key you can use with this package.

The MSCS Text Analytics REST API

Microsoft Cognitive Services – formerly known as Project Oxford – are a set of APIs, SDKs and services that developers can use to add AI features to their apps. Those features include emotion and video detection; facial, speech and vision recognition; as well as speech and NLP.

The Text Analytics REST API provides tools for NLP and is documented at https://www.microsoft.com/cognitive-services/en-us/text-analytics/documentation. This API supports the following operations:

- Sentiment analysis - Is a sentence or document generally positive or negative?
- Topic detection - What's being discussed across a list of documents/reviews/articles?
- Language detection - What language is a document written in?
- Key talking points extraction - What's being discussed in a single document?

mscstexta4r Functions

The following **mscstexta4r** core functions are used to wrap the MSCS Text Analytics REST API:

- Sentiment analysis - `textaSentiment` function
- Topic detection - `textaDetectTopics` and `textaDetectTopicsStatus` functions
- Language detection - `textaDetectLanguages` function
- Extraction of key talking points - `textaKeyPhrases` function

The `textaInit` configuration function is used to set the REST API URL and the private API key. It needs to be called *only once*, after package load, or the core functions will not work properly.

Prerequisites

To use the **mscstexta4r** R package, you **MUST** have a valid account with Microsoft Cognitive Services (see https://www.microsoft.com/cognitive-services/en-us/pricing for details). Once you have an account, Microsoft will provide you with an API key listed under your subscriptions. After you've configured **mscstexta4r** with your API key (as explained in the next section), you will be able to call the Text Analytics REST API from R, up to your maximum number of transactions per month and per minute.

Package Loading and Configuration

After loading the **mscstexta4r** package with the `library()` function, you must call the `textaInit` before you can call any of the core **mscstexta4r** functions.

The `textaInit` configuration function will first check to see if the variable `MSCS_TEXTANALYTICS_CONFIG_FILE` exists in the system environment. If it does, the package will use that as the path to the configuration file.

If `MSCS_TEXTANALYTICS_CONFIG_FILE` doesn't exist, it will look for the file `.mscskeys.json` in the current user's home directory (that's `~/.mscskeys.json` on Linux, and something like `C:/Users/Phil/Documents/.mscskeys.json` on Windows). If the file is found, the package will load the API key and URL from it.

If using a file, please make sure it has the following structure:

```
{
 "textanalyticsurl": "https://westus.api.cognitive.microsoft.com/texta/analytics/v2.0/",
  "textanalyticskey": "...MSCS Text Analytics API key goes here..."
}
```

If no configuration file is found, `textaInit` will attempt to pick up its configuration information from two Sys env variables instead:

`MSCS_TEXTANALYTICS_URL` - the URL for the Text Analytics REST API.

`MSCS_TEXTANALYTICS_KEY` - your personal Text Analytics REST API key.

Synchronous vs Asynchronous Execution

All but **ONE** core text analytics functions execute exclusively in synchronous mode: `textaDetectTopics` is the only function that can be executed either synchronously or asynchronously. Why? Because topic detection is typically a "batch" operation meant to be performed on thousands of related documents (product reviews, research articles, etc.).

What's the difference?

When `textaDetectTopics` executes synchronously, you must wait for it to finish before you can move on to the next task. When `textaDetectTopics` executes asynchronously, you can move on to something else before topic detection has completed. In the latter case, you will need to call `textaDetectTopicsStatus` periodically yourself until the Microsoft Cognitive Services server complete topic detection and results become available.

When to run which mode?

If you're performing topic detection in batch mode (from an R script), we recommend using the `textaDetectTopics` function in synchronous mode, in which case it will return only after topic detection has completed.

IMPORTANT NOTE: If you're calling `textaDetectTopics` in synchronous mode within the R console REPL (interactive mode), it will appear as if the console has hanged. This is *EXPECTED*. The function hasn't crashed. It is simply in "sleep mode", activating itself periodically and then going back to sleep, until the results have become available. In sleep mode, even though it appears "stuck", `textaDetectTopics` doesn't use any CPU resources. While the function is operating in sleep mode, you *WILL NOT* be able to use the console before the function completes. If you need to operate the console while topic detection is being performed by the Microsoft Cognitive services servers, you should call `textaDetectTopics` in asynchronous mode and then call `textaDetectTopicsStatus` yourself repeteadly afterwards, until results are available.

S3 Objects of the Classes `texta` and `textatopics`

The sentiment analysis, language detection, and key talking points extraction functions of the **mscstexta4r** package return S3 objects of the class `texta`. The `texta` object exposes results collected in a single dataframe, the REST API JSON response, and the original HTTP request.

The functions `textaDetectTopics` returns a S3 object of the class `textatopics`. The `textatopics` object exposes formatted results using several dataframes (documents and their IDs, topics and their IDs, which topics are assigned to which documents), the REST API JSON response (should you care), and the HTTP request (mostly for debugging purposes).'

Error Handling

The MSCS Text Analytics API is a REST API. HTTP requests over a network and the Internet can fail. Because of congestion, because the web site is down for maintenance, because of firewall configuration issues, etc. There are many possible points of failure.

The API can also fail if you've exhausted your call volume quota or are exceeding the API calls rate limit. Unfortunately, MSCS does not expose an API you can query to check if you're about to exceed your quota for instance. The only way you'll know for sure is by looking at the error code returned after an API call has failed.

To help with error handling, we recommend the systematic use of `tryCatch()` when calling **mscstexta4r**'s core functions. Its mechanism may appear a bit daunting at first, but it is well documented at `http://www.inside-r.org/r-doc/base/signalCondition`. We use it in many of the code examples.

Author(s)

Phil Ferriere <pferriere@hotmail.com>

texta *The* texta *object*

Description

The texta object exposes formatted results, the REST API JSON response, and the HTTP request:

- result the results in data.frame format
- json the REST API JSON response
- request the HTTP request

Author(s)

Phil Ferriere <pferriere@hotmail.com>

See Also

Other res: textatopics

textaDetectLanguages *Detects the languages used in documents.*

Description

This function returns the language detected in a sentence or documents along with a confidence score between 0 and 1. A scores equal to 1 indicates 100

Internally, this function invokes the Microsoft Cognitive Services Text Analytics REST API documented at https://www.microsoft.com/cognitive-services/en-us/text-analytics/documentation.

You MUST have a valid Microsoft Cognitive Services account and an API key for this function to work properly. See https://www.microsoft.com/cognitive-services/en-us/pricing for details.

Usage

```
textaDetectLanguages(documents, numberOfLanguagesToDetect = 1L)
```

Arguments

documents	(character vector) Vector of sentences or documents on which to perform language detection.
numberOfLanguagesToDetect	(integer) Number of languages to detect. Set to 1 by default. Use a higher value if individual documents contain a mix of languages.

Value

An S3 object of the class texta. The results are stored in the results dataframe inside this object. The dataframe contains the original sentences or documents, the name of the detected language, the ISO 639-1 code of the detected language, and a confidence score. If an error occurred during processing, the dataframe will also have an error column that describes the error.

Author(s)

Phil Ferriere <pferriere@hotmail.com>

Examples

```
## Not run:

 docsText <- c(
   "The Louvre or the Louvre Museum is the world's largest museum.",
   "Le musee du Louvre est un musee d'art et d'antiquites situe au centre de Paris.",
   "El Museo del Louvre es el museo nacional de Francia.",
   "Il Museo del Louvre a Parigi, in Francia, e uno dei piu celebri musei del mondo.",
   "Der Louvre ist ein Museum in Paris."
 )

 tryCatch({

   # Detect languages used in documents
   docsLanguage <- textaDetectLanguages(
     documents = docsText,           # Input sentences or documents
     numberOfLanguagesToDetect = 1L  # Number of languages to detect
   )

   # Class and structure of docsLanguage
   class(docsLanguage)
   #> [1] "texta"
   str(docsLanguage, max.level = 1)
   #> List of 3
   #>  $ results:'data.frame': 5 obs. of  4 variables:
  #> $ json  : chr "{\"documents\":[{\"id\":\"B6e4C\",\"detectedLanguages\": __truncated__ }]}
   #>  $ request:List of 7
   #>   ..- attr(*, "class")= chr "request"
   #>  - attr(*, "class")= chr "texta"

   # Print results
   docsLanguage
  #> texta [https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/lan __truncated__ ]
   #>
   #> ------------------------------------------------------------
   #>             text               name    iso6391Name   score
   #> --------------------------- ------- ------------- -------
   #>   The Louvre or the Louvre    English      en          1
   #> Museum is the world's largest
   #>            museum.
   #>
```

```
#>   Le musee du Louvre est un    French      fr          1
#>  musee d'art et d'antiquites
#>   situe au centre de Paris.
#>
#>   El Museo del Louvre es el   Spanish      es          1
#>  museo nacional de Francia.
#>
#> Il Museo del Louvre a Parigi, Italian      it          1
#>   in Francia, e uno dei piu
#>   celebri musei del mondo.
#>
#>  Der Louvre ist ein Museum in  German      de          1
#>            Paris.
#> ------------------------------------------------------------

}, error = function(err) {

  # Print error
  geterrmessage()

})

## End(Not run)
```

`textaDetectTopics` *Detects the top topics in a group of text documents.*

Description

This function returns the top detected topics for a list of submitted text documents. A topic is identified with a key phrase, which can be one or more related words. At least 100 text documents must be submitted, however this API is designed to detect topics across hundreds to thousands of documents. For best performance, limit each document to a short, human written text paragraph such as review, conversation or user feedback.

English is the only language supported at this time.

You can provide a list of stop words to control which words or documents are filtered out. You can also supply a list of topics to exclude from the response. Finally, you can also provide min/max word frequency count thresholds to exclude rare/ubiquitous document topics.

We recommend using the `textaDetectTopics` function in synchronous mode, in which case it will return only after topic detection has completed. If you decide to call this function in asynchronous mode, you will need to call the `textaDetectTopicsStatus` function periodically yourself until the Microsoft Cognitive Services server complete topic detection and results become available.

IMPORTANT NOTE: If you're calling `textaDetectTopics` in synchronous mode within the R console REPL (interactive mode), it will appear as if the console has hanged. This is *EXPECTED*. The function hasn't crashed. It is simply in "sleep mode", activating itself periodically and then going back to sleep, until the results have become available. In sleep mode, even though it appears "stuck", `textaDetectTopics` dodesn't use any CPU resources. While

the function is operating in sleep mode, you *WILL NOT* be able to use the console until the function completes. If need to operate the console while topic detection is being performed by the Microsoft Cognitive services servers, you should call `textaDetectTopics` in asynchronous mode and then call `textaDetectTopicsStatus` yourself repeteadly afterwards, until results are available.

Note that one transaction is charged per text document submitted.

Internally, this function invokes the Microsoft Cognitive Services Text Analytics REST API documented at https://www.microsoft.com/cognitive-services/en-us/text-analytics/documentation.

You MUST have a valid Microsoft Cognitive Services account and an API key for this function to work properly. See https://www.microsoft.com/cognitive-services/en-us/pricing for details.

Usage

```
textaDetectTopics(documents, stopWords = NULL, topicsToExclude = NULL,
  minDocumentsPerWord = NULL, maxDocumentsPerWord = NULL,
  resultsPollInterval = 30L, resultsTimeout = 1200L, verbose = FALSE)
```

Arguments

`documents`	(character vector) Vector of sentences or documents on which to perform topic detection. At least 100 text documents must be submitted. English is the only language supported at this time.
`stopWords`	(character vector) Vector of stop words to ignore while performing topic detection (optional)
`topicsToExclude`	(character vector) Vector of topics to exclude from the response (optional)
`minDocumentsPerWord`	(integer) Words that occur in less than this many documents are ignored. Use this parameter to help exclude rare document topics. Omit to let the service choose appropriate value. (optional)
`maxDocumentsPerWord`	(integer) Words that occur in more than this many documents are ignored. Use this parameter to help exclude ubiquitous document topics. Omit to let the service choose appropriate value. (optional)
`resultsPollInterval`	(integer) Interval (in seconds) at which this function will query the Microsoft Cognitive Services servers for results (optional, default: 30L). If set to 0L, this function will return immediately and you will have to call `textaDetectTopicsStatus` periodically to collect results. If set to a non-zero integer value, this function will only return after all results have been collected. It does so by repeatedly calling `textaDetectTopicsStatus` on its own until topic detection has completed. In the latter case, you do not need to call `textaDetectTopicsStatus`.
`resultsTimeout`	(integer) Interval (in seconds) at which point this function will give up and stop querying the Microsoft Cognitive Services servers for results (optional, default: 1200L). As soon as all results are available, this function will return them to the caller. If the Microsoft Cognitive Services servers within resultsTimeout

seconds, this function will stop polling the servers and return the most current results.

`verbose` (logical) If set to TRUE, print every poll status to stdout.

Value

An S3 object of the class `textatopics`. The results are stored in the `results` dataframes inside this object. See `textatopics` for details. In the synchronous case (i.e., the function only returns after completion), the dataframes contain the documents, the topics, and which topics are assigned to which documents. In the asynchonous case (i.e., the function returns immediately), the dataframes contain the documents, their unique identifiers, their current operation status code, but they don't contain the topics yet, nor their assignments. To get the topics and their assignments, you must call `textaDetectTopicsStatus` until the Microsoft Services servers have completed topic detection.

Author(s)

Phil Ferriere <pferriere@hotmail.com>

Examples

```
## Not run:
 load("./data/yelpChineseRestaurantReviews.rda")
 set.seed(1234)
 documents <- sample(yelpChReviews$text, 1000)

 tryCatch({

   # Detect top topics in group of documents
   topics <- textaDetectTopics(
     documents,                  # At least 100 documents (English only)
     stopWords = NULL,           # Stop word list (optional)
     topicsToExclude = NULL,     # Topics to exclude (optional)
     minDocumentsPerWord = NULL, # Threshold to exclude rare topics (optional)
     maxDocumentsPerWord = NULL, # Threshold to exclude ubiquitous topics (optional)
     resultsPollInterval = 30L,  # Poll interval (in s, default:30s, use 0L for async)
     resultsTimeout = 1200L,     # Give up timeout (in s, default: 1200s = 20mn)
     verbose = TRUE              # If set to TRUE, print every poll status to stdout
   )

   # Class and structure of topics
   class(topics)
   #> [1] "textatopics"

   str(topics, max.level = 1)
   #> List of 8
   #> $ status          : chr "Succeeded"
   #> $ operationId     : chr "30334a3e1e28406a80566bb76ff04884"
   #> $ operationType   : chr "topics"
   #> $ documents       :'data.frame':  1000 obs. of  2 variables:
   #> $ topics          :'data.frame':  71 obs. of  3 variables:
   #> $ topicAssignments:'data.frame':  502 obs. of  3 variables:
```

```
#> $ json         : chr "{\"status\":\"Succeeded\",\"createdDateTime\": __truncated__ }
 #> $ request         :List of 7
 #> ..- attr(*, "class")= chr "request"
 #> - attr(*, "class")= chr "textatopics"

 # Print results
 topics
#> textatopics [https://westus.api.cognitive.microsoft.com/text/analytics/ __truncated__ ]
 #> status: Succeeded
 #> operationId: 30334a3e1e28406a80566bb76ff04884
 #> operationType: topics
 #> topics (first 20):
 #> ------------------------
 #>    keyPhrase      score
 #> ---------------- -------
 #>     portions       35
 #>   noodle soup      30
 #>    vegetables      20
 #>       tofu         19
 #>      garlic        17
 #>     Eggplant       15
 #>       Pad          15
 #>      combo         13
 #> Beef Noodle Soup   13
 #>      House         12
 #>      entree        12
 #>     wontons        12
 #>     Pei Wei        12
 #>  mongolian beef    11
 #>       crab         11
 #>      Panda         11
 #>       bean         10
 #>    dumplings        9
 #>     veggies         9
 #>      decor          9
 #> ------------------------

}, error = function(err) {

  # Print error
  geterrmessage()

})

## End(Not run)
```

`textaDetectTopicsStatus` *Retrieves the status of a topic detection operation submitted for processing.*

Description

This function retrieves the status of an asynchronous topic detection operation previously submitted for processing. If the operation has reached a 'Succeeded' state, this function will also return the results.

Internally, this function invokes the Microsoft Cognitive Services Text Analytics REST API documented at https://www.microsoft.com/cognitive-services/en-us/text-analytics/documentation.

You MUST have a valid Microsoft Cognitive Services account and an API key for this function to work properly. See https://www.microsoft.com/cognitive-services/en-us/pricing for details.

Usage

```
textaDetectTopicsStatus(operation, verbose = FALSE)
```

Arguments

operation	(textatopics) textatopics S3 object returned by the original call to textaDetectTopics.
verbose	(logical) If set to TRUE, print poll status to stdout.

Value

An S3 object of the class textatopics with the results of the topic detection operation. See textatopics for details.

Author(s)

Phil Ferriere <pferriere@hotmail.com>

Examples

```
## Not run:
 load("./data/yelpChineseRestaurantReviews.rda")
 set.seed(1234)
 documents <- sample(yelpChReviews$text, 1000)

 tryCatch({

   # Start async topic detection
   operation <- textaDetectTopics(
     documents,                  # At least 100 docs/sentences
     stopWords = NULL,           # Stop word list (optional)
     topicsToExclude = NULL,     # Topics to exclude (optional)
     minDocumentsPerWord = NULL, # Threshold to exclude rare topics (optional)
     maxDocumentsPerWord = NULL, # Threshold to exclude ubiquitous topics (optional)
     resultsPollInterval = 0L    # Poll interval (in s, default: 30s, use 0L for async)
   )

   # Poll the servers until the work completes or until we time out
   resultsPollInterval <- 60L
   resultsTimeout <- 1200L
```

```
startTime <- Sys.time()
endTime <- startTime + resultsTimeout

while (Sys.time() <= endTime) {
  sleepTime <- startTime + resultsPollInterval - Sys.time()
  if (sleepTime > 0)
    Sys.sleep(sleepTime)
  startTime <- Sys.time()

  # Poll for results
  topics <- textaDetectTopicsStatus(operation)
  if (topics$status != "NotStarted" && topics$status != "Running")
    break;
}

# Class and structure of topics
class(topics)
#> [1] "textatopics"

str(topics, max.level = 1)
#> List of 8
#> $ status          : chr "Succeeded"
#> $ operationId     : chr "30334a3e1e28406a80566bb76ff04884"
#> $ operationType   : chr "topics"
#> $ documents       :'data.frame':  1000 obs. of  2 variables:
#> $ topics          :'data.frame':  71 obs. of  3 variables:
#> $ topicAssignments:'data.frame':  502 obs. of  3 variables:
#> $ json         : chr "{\"status\":\"Succeeded\",\"createdDateTime\": __truncated__ }
#> $ request         :List of 7
#> ..- attr(*, "class")= chr "request"
#> - attr(*, "class")= chr "textatopics"

# Print results
topics
#> textatopics [https://westus.api.cognitive.microsoft.com/text/analytics/ __truncated__ ]
#> status: Succeeded
#> operationId: 30334a3e1e28406a80566bb76ff04884
#> operationType: topics
#> topics (first 20):
#> ------------------------
#>    keyPhrase      score
#> ---------------- -------
#>     portions       35
#>   noodle soup      30
#>    vegetables      20
#>       tofu         19
#>      garlic        17
#>     Eggplant       15
#>       Pad          15
#>      combo         13
#> Beef Noodle Soup   13
#>      House         12
#>      entree        12
```

```
#>     wontons        12
#>     Pei Wei        12
#>  mongolian beef    11
#>       crab         11
#>      Panda         11
#>       bean         10
#>    dumplings        9
#>     veggies         9
#>      decor          9
#> ------------------------

}, error = function(err) {

  # Print error
  geterrmessage()

})

## End(Not run)
```

`textaInit` *Initializes the* **mscstexta4r** *package.*

Description

This function initializes the Microsoft Cognitive Services Text Analytics REST API key and URL by reading them either from a configuration file or environment variables.

This function **MUST** be called right after package load and before calling any **mscstexta4r** core functions, or these functions will fail.

The `textaInit` configuration function will first check to see if the variable `MSCS_TEXTANALYTICS_CONFIG_FILE` exists in the system environment. If it does, the package will use that as the path to the configuration file.

If `MSCS_TEXTANALYTICS_CONFIG_FILE` doesn't exist, it will look for the file `.mscskeys.json` in the current user's home directory (that's `~/.mscskeys.json` on Linux, and something like `C:/Users/Phil/Documents/.mscskeys.json` on Windows). If the file is found, the package will load the API key and URL from it.

If using a file, please make sure it has the following structure:

```
{
 "textanalyticsurl": "https://westus.api.cognitive.microsoft.com/texta/analytics/v2.0/",
  "textanalyticskey": "...MSCS Text Analytics API key goes here..."
}
```

If no configuration file is found, `textaInit` will attempt to pick up its configuration information from two Sys env variables instead:

`MSCS_TEXTANALYTICS_URL` - the URL for the Text Analytics REST API.

`MSCS_TEXTANALYTICS_KEY` - your personal Text Analytics REST API key.

`textaInit` needs to be called *only once*, after package load.

Usage

```
textaInit()
```

Author(s)

Phil Ferriere <pferriere@hotmail.com>

Examples

```
## Not run:
 textaInit()

## End(Not run)
```

---

textaKeyPhrases *Returns the key talking points in sentences or documents.*

Description

This function returns the the key talking points in a list of sentences or documents. The following languages are currently supported: English, German, Spanish and Japanese.

Internally, this function invokes the Microsoft Cognitive Services Text Analytics REST API documented at https://www.microsoft.com/cognitive-services/en-us/text-analytics/documentation.

You MUST have a valid Microsoft Cognitive Services account and an API key for this function to work properly. See https://www.microsoft.com/cognitive-services/en-us/pricing for details.

Usage

```
textaKeyPhrases(documents, languages = rep("en", length(documents)))
```

Arguments

documents	(character vector) Vector of sentences or documents for which to extract key talking points.
languages	(character vector) Languages of the sentences or documents, supported values: "en"(English, default), "de"(German), "es"(Spanish), "fr"(French), "ja"(Japanese)

Value

An S3 object of the class texta. The results are stored in the results dataframe inside this object. The dataframe contains the original sentences or documents and their key talking points. If an error occurred during processing, the dataframe will also have an error column that describes the error.

Author(s)

Phil Ferriere <pferriere@hotmail.com>

Examples

```
## Not run:

docsText <- c(
  "Loved the food, service and atmosphere! We'll definitely be back.",
  "Very good food, reasonable prices, excellent service.",
  "It was a great restaurant.",
  "If steak is what you want, this is the place.",
  "The atmosphere is pretty bad but the food is quite good.",
  "The food is quite good but the atmosphere is pretty bad.",
  "I'm not sure I would come back to this restaurant.",
  "The food wasn't very good.",
  "While the food was good the service was a disappointment.",
  "I was very disappointed with both the service and my entree."
)
docsLanguage <- rep("en", length(docsText))

tryCatch({

  # Get key talking points in documents
  docsKeyPhrases <- textaKeyPhrases(
    documents = docsText,    # Input sentences or documents
    languages = docsLanguage
    # "en"(English, default)|"de"(German)|"es"(Spanish)|"fr"(French)|"ja"(Japanese)
  )

  # Class and structure of docsKeyPhrases
  class(docsKeyPhrases)
  #> [1] "texta"
  str(docsKeyPhrases, max.level = 1)
  #> List of 3
  #>  $ results:'data.frame': 10 obs. of  2 variables:
 #> $ json  : chr "{\"documents\":[{\"keyPhrases\":[\"atmosphere\",\"food\", __truncated__ ]}]}
  #>  $ request:List of 7
  #>   ..- attr(*, "class")= chr "request"
  #>  - attr(*, "class")= chr "texta"

  # Print results
  docsKeyPhrases
  #> texta [https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/keyPhrases]
  #>
  #> ------------------------------------------------------------
  #>              text                       keyPhrases
  #> ------------------------------ --------------------------
  #>  Loved the food, service and    atmosphere, food, service
  #>  atmosphere! We'll definitely
  #>            be back.
  #>
  #>   Very good food, reasonable   reasonable prices, good food
  #>   prices, excellent service.
  #>
  #>   It was a great restaurant.         great restaurant
```

```
#>
#>   If steak is what you want,           steak, place
#>       this is the place.
#>
#>  The atmosphere is pretty bad        atmosphere, food
#>  but the food is quite good.
#>
#>  The food is quite good but the       food, atmosphere
#>   atmosphere is pretty bad.
#>
#> I'm not sure I would come back          restaurant
#>      to this restaurant.
#>
#>   The food wasn't very good.               food
#>
#>  While the food was good the          service, food
#> service was a disappointment.
#>
#>  I was very disappointed with        service, entree
#>    both the service and my
#>            entree.
#> ------------------------------------------------------------

}, error = function(err) {

  # Print error
  geterrmessage()

})

## End(Not run)
```

`textaSentiment` *Assesses the sentiment of sentences or documents.*

Description

This function returns a numeric score between 0 and 1 with scores close to 1 indicating positive sentiment and scores close to 0 indicating negative sentiment.

Sentiment score is generated using classification techniques. The input features of the classifier include n-grams, features generated from part-of-speech tags, and word embeddings. English, French, Spanish and Portuguese text are supported.

Internally, this function invokes the Microsoft Cognitive Services Text Analytics REST API documented at https://www.microsoft.com/cognitive-services/en-us/text-analytics/documentation.

You MUST have a valid Microsoft Cognitive Services account and an API key for this function to work properly. See https://www.microsoft.com/cognitive-services/en-us/pricing for details.

Usage

```
textaSentiment(documents, languages = rep("en", length(documents)))
```

Arguments

documents	(character vector) Vector of sentences or documents for which to assess sentiment.
languages	(character vector) Languages of the sentences or documents, supported values: "en"(English, default), "es"(Spanish), "fr"(French), "pt"(Portuguese)

Value

An S3 object of the class `texta`. The results are stored in the `results` dataframe inside this object. The dataframe contains the original sentences or documents and their sentiment score. If an error occurred during processing, the dataframe will also have an `error` column that describes the error.

Author(s)

Phil Ferriere <pferriere@hotmail.com>

Examples

```
## Not run:

 docsText <- c(
   "Loved the food, service and atmosphere! We'll definitely be back.",
   "Very good food, reasonable prices, excellent service.",
   "It was a great restaurant.",
   "If steak is what you want, this is the place.",
   "The atmosphere is pretty bad but the food is quite good.",
   "The food is quite good but the atmosphere is pretty bad.",
   "I'm not sure I would come back to this restaurant.",
   "The food wasn't very good.",
   "While the food was good the service was a disappointment.",
   "I was very disappointed with both the service and my entree."
 )
 docsLanguage <- rep("en", length(docsText))

 tryCatch({

   # Perform sentiment analysis
   docsSentiment <- textaSentiment(
     documents = docsText,    # Input sentences or documents
     languages = docsLanguage
     # "en"(English, default)|"es"(Spanish)|"fr"(French)|"pt"(Portuguese)
   )

   # Class and structure of docsSentiment
   class(docsSentiment)
   #> [1] "texta"
   str(docsSentiment, max.level = 1)
```

```
  #> List of 3
  #>  $ results:'data.frame': 10 obs. of  2 variables:
 #> $ json  : chr "{\"documents\":[{\"score\":0.9903013,\"id\":\"hDgKc\", __truncated__ }]}
  #>  $ request:List of 7
  #>   ..- attr(*, "class")= chr "request"
  #>  - attr(*, "class")= chr "texta"

  # Print results
  docsSentiment
  #> texta [https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/sentiment]
  #>
  #> --------------------------------------
  #>              text               score
  #> ------------------------------ -------
  #>  Loved the food, service and    0.9847
  #>  atmosphere! We'll definitely
  #>            be back.
  #>
  #>  Very good food, reasonable     0.9831
  #>  prices, excellent service.
  #>
  #>  It was a great restaurant.     0.9306
  #>
  #>  If steak is what you want,     0.8014
  #>      this is the place.
  #>
  #> The atmosphere is pretty bad    0.4998
  #>  but the food is quite good.
  #>
  #> The food is quite good but the   0.475
  #>   atmosphere is pretty bad.
  #>
  #> I'm not sure I would come back  0.2857
  #>      to this restaurant.
  #>
  #>   The food wasn't very good.    0.1877
  #>
  #>  While the food was good the   0.08727
  #> service was a disappointment.
  #>
  #>  I was very disappointed with  0.01877
  #>    both the service and my
  #>            entree.
  #> --------------------------------------

 }, error = function(err) {

  # Print error
  geterrmessage()

 })

## End(Not run)
```

`textatopics` *The* `textatopics` *object*

Description

The `textatopics` object exposes formatted results for the `textaDetectTopics` API, this REST API's JSON response, and the HTTP request:

- `status` the operation's current status ("NotStarted"|"Running"|"Succeeded"|"Failed")
- `documents` a `data.frame` with the documents and a unique string ID for each
- `topics` a `data.frame` with the identified topics, a unique string ID for each, and a prevalence score for each topic (count of documents assigned to topic)
- `topicAssignments` a `data.frame` with all the topics (identified by their topic ID) assigned to each document (identified by their document ID), and a distance score for each topic assignment (between 0 and 1; the lower the distance score the stronger the topic affiliation)
- `json` the REST API JSON response
- `request` the HTTP request

Author(s)

Phil Ferriere <pferriere@hotmail.com>

See Also

Other res: `texta`

Index