Package 'googleLanguageR'

July 22, 2025

Title Call Google's 'Natural Language' API, 'Cloud Translation' API, 'Cloud Speech' API and 'Cloud Text-to-Speech' API

Version 0.3.0

Description Call 'Google Cloud' machine learning APIs for text and speech tasks. Call the 'Cloud Translation' API <https://cloud.google.com/translate/> for detection and translation of text, the 'Natural Language' API <https://cloud.google.com/natural-language/> to analyse text for sentiment, entities or syntax, the 'Cloud Speech' API <https://cloud.google.com/speech/> to transcribe sound files to text and the 'Cloud Text-to-Speech' API <https://cloud.google.com/text-to-speech/> to turn text into sound files.

URL http://code.markedmondson.me/googleLanguageR/, https://github.com/ropensci/googleLanguageR, https://docs.ropensci.org/googleLanguageR/

BugReports https://github.com/ropensci/googleLanguageR/issues

Depends R (>= 3.3)

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

VignetteBuilder knitr

Imports assertthat, base64enc, googleAuthR (>= 1.1.1), jsonlite, magrittr, purrr (>= 0.2.4), stats, tibble, utils

Suggests cld2, testthat, knitr, rmarkdown, rvest, shiny, shinyjs, stringdist, tidyr, tuneR, xml2

NeedsCompilation no

Author Mark Edmondson [aut, cre], John Muschelli [ctb], Neal Richardson [rev] (Neal reviewed the package for ropensci, see <https://github.com/ropensci/onboarding/issues/127>), Julia Gustavsen [rev] (Julia reviewed the package for ropensci, see <https://github.com/ropensci/onboarding/issues/127>)

Maintainer Mark Edmondson <r@sunholo.com>

Repository CRAN

Date/Publication 2020-04-19 13:40:02 UTC

Contents

`gl_auth` *Authenticate with Google language API services*

Description

Authenticate with Google language API services

Usage

```
gl_auth(json_file)

gl_auto_auth(...)
```

Arguments

`json_file`	Authentication json file you have downloaded from your Google Project
`...`	additional argument to pass to `gar_attach_auto_auth`.

Details

The best way to authenticate is to use an environment argument pointing at your authentication file.

Set the file location of your download Google Project JSON file in a `GL_AUTH` argument

Then, when you load the library you should auto-authenticate

However, you can authenticate directly using this function pointing at your JSON auth file.

Examples

```
## Not run:
library(googleLanguageR)
gl_auth("location_of_json_file.json")

## End(Not run)

## Not run:
library(googleLanguageR)
gl_auto_auth()
gl_auto_auth(environment_var = "GAR_AUTH_FILE")

## End(Not run)
```

`gl_nlp` *Perform Natural Language Analysis*

Description

Analyse text entities, sentiment, syntax and categorisation using the Google Natural Language API

Usage

```
gl_nlp(
  string,
 nlp_type = c("annotateText", "analyzeEntities", "analyzeSentiment", "analyzeSyntax",
    "analyzeEntitySentiment", "classifyText"),
  type = c("PLAIN_TEXT", "HTML"),
  language = c("en", "zh", "zh-Hant", "fr", "de", "it", "ja", "ko", "pt", "es"),
  encodingType = c("UTF8", "UTF16", "UTF32", "NONE")
)
```

Arguments

`string`	A vector of text to detect language for, or Google Cloud Storage URI(s)
`nlp_type`	The type of Natural Language Analysis to perform. The default `annotateText` will perform all features in one call.
`type`	Whether input text is plain text or a HTML page
`language`	Language of source, must be supported by API.
`encodingType`	Text encoding that the caller uses to process the output

Details

`string` can be a character vector, or a location of a file content on Google cloud Storage. This URI must be of the form `gs://bucket_name/object_name`

Encoding type can usually be left at default `UTF8`. Read more here

The current language support is available here

Value

A list of the following objects, if those fields are asked for via `nlp_type`:

- sentences - Sentences in the input document
- tokens - Tokens, along with their syntactic information, in the input document
- entities - Entities, along with their semantic information, in the input document
- documentSentiment - The overall sentiment for the document
- classifyText -Classification of the document
- language - The language of the text, which will be the same as the language specified in the request or, if not specified, the automatically-detected language
- text - The original text passed into the API. `NA` if not passed due to being zero-length etc.

See Also

https://cloud.google.com/natural-language/docs/reference/rest/v1/documents

Examples

```
## Not run:

text <- "to administer medicince to animals is frequently a very difficult matter,
  and yet sometimes it's necessary to do so"
nlp <- gl_nlp(text)

nlp$sentences

nlp$tokens

nlp$entities

nlp$documentSentiment

## vectorised input
texts <- c("The cat sat one the mat", "oh no it didn't you fool")
nlp_results <- gl_nlp(texts)




## End(Not run)
```

gl_speech *Call Google Speech API*

Description

Turn audio into text

Usage

```
gl_speech(
  audio_source,
  encoding = c("LINEAR16", "FLAC", "MULAW", "AMR", "AMR_WB", "OGG_OPUS",
    "SPEEX_WITH_HEADER_BYTE"),
  sampleRateHertz = NULL,
  languageCode = "en-US",
  maxAlternatives = 1L,
  profanityFilter = FALSE,
  speechContexts = NULL,
  asynch = FALSE,
  customConfig = NULL
)
```

Arguments

audio_source	File location of audio data, or Google Cloud Storage URI
encoding	Encoding of audio data sent
sampleRateHertz	Sample rate in Hertz of audio data. Valid values 8000-48000. Optimal and default if left NULL is 16000
languageCode	Language of the supplied audio as a BCP-47 language tag
maxAlternatives	Maximum number of recognition hypotheses to be returned. 0-30
profanityFilter	If TRUE will attempt to filter out profanities
speechContexts	An optional character vector of context to assist the speech recognition
asynch	If your audio_source is greater than 60 seconds, set this to TRUE to return an asynchronous call
customConfig	[optional] A RecognitionConfig object that will be converted from a list to JSON via toJSON - see RecognitionConfig documentation. The languageCode will be taken from this functions arguments if not present since it is required.

Details

Google Cloud Speech API enables developers to convert audio to text by applying powerful neural network models in an easy to use API. The API recognizes over 80 languages and variants, to support your global user base. You can transcribe the text of users dictating to an application's microphone, enable command-and-control through voice, or transcribe audio files, among many other use cases. Recognize audio uploaded in the request, and integrate with your audio storage on Google Cloud Storage, by using the same technology Google uses to power its own products.

Value

A list of two tibbles: `$transcript`, a tibble of the `transcript` with a `confidence`; `$timings`, a tibble that contains `startTime`, `endTime` per `word`. If maxAlternatives is greater than 1, then the transcript will return near-duplicate rows with other interpretations of the text. If `asynch` is TRUE, then an operation you will need to pass to gl_speech_op to get the finished result.

AudioEncoding

Audio encoding of the data sent in the audio message. All encodings support only 1 channel (mono) audio. Only FLAC and WAV include a header that describes the bytes of audio that follow the header. The other encodings are raw audio bytes with no header. For best results, the audio source should be captured and transmitted using a lossless encoding (FLAC or LINEAR16). Recognition accuracy may be reduced if lossy codecs, which include the other codecs listed in this section, are used to capture or transmit the audio, particularly if background noise is present.

Read more on audio encodings here `https://cloud.google.com/speech/docs/encoding`

WordInfo

`startTime` - Time offset relative to the beginning of the audio, and corresponding to the start of the spoken word.

`endTime` - Time offset relative to the beginning of the audio, and corresponding to the end of the spoken word.

`word` - The word corresponding to this set of information.

See Also

`https://cloud.google.com/speech/reference/rest/v1/speech/recognize`

Examples

```
## Not run:

test_audio <- system.file("woman1_wb.wav", package = "googleLanguageR")
result <- gl_speech(test_audio)

result$transcript
result$timings

result2 <- gl_speech(test_audio, maxAlternatives = 2L)
result2$transcript
```

```
result_brit <- gl_speech(test_audio, languageCode = "en-GB")


## make an asynchronous API request (mandatory for sound files over 60 seconds)
asynch <- gl_speech(test_audio, asynch = TRUE)

## Send to gl_speech_op() for status or finished result
gl_speech_op(asynch)

## Upload to GCS bucket for long files > 60 seconds
test_gcs <- "gs://mark-edmondson-public-files/googleLanguageR/a-dream-mono.wav"
gcs <- gl_speech(test_gcs, sampleRateHertz = 44100L, asynch = TRUE)
gl_speech_op(gcs)

## Use a custom configuration
my_config <- list(encoding = "LINEAR16",
                  diarizationConfig = list(
                    enableSpeakerDiarization = TRUE,
                    minSpeakerCount = 2,
                    maxSpeakCount = 3
                    ))

# languageCode is required, so will be added if not in your custom config
gl_speech(my_audio, languageCode = "en-US", customConfig = my_config)


## End(Not run)
```

`gl_speech_op` *Get a speech operation*

Description

For asynchronous calls of audio over 60 seconds, this returns the finished job

Usage

```
gl_speech_op(operation = .Last.value)
```

Arguments

`operation`	A speech operation object from gl_speech when `asynch = TRUE`

Value

If the operation is still running, another operation object. If done, the result as per gl_speech

See Also

gl_speech

Examples

```
## Not run:

test_audio <- system.file("woman1_wb.wav", package = "googleLanguageR")

## make an asynchronous API request (mandatory for sound files over 60 seconds)
asynch <- gl_speech(test_audio, asynch = TRUE)

## Send to gl_speech_op() for status or finished result
gl_speech_op(asynch)


## End(Not run)
```

---

gl_talk *Perform text to speech*

Description

Synthesizes speech synchronously: receive results after all text input has been processed.

Usage

```
gl_talk(
  input,
  output = "output.wav",
  languageCode = "en",
  gender = c("SSML_VOICE_GENDER_UNSPECIFIED", "MALE", "FEMALE", "NEUTRAL"),
  name = NULL,
  audioEncoding = c("LINEAR16", "MP3", "OGG_OPUS"),
  speakingRate = 1,
  pitch = 0,
  volumeGainDb = 0,
  sampleRateHertz = NULL,
  inputType = c("text", "ssml"),
  effectsProfileIds = NULL
)
```

Arguments

input	The text to turn into speech
output	Where to save the speech audio file

languageCode	The language of the voice as a BCP-47 language code
gender	The gender of the voice, if available
name	Name of the voice, see list via gl_talk_languages for supported voices. Set to NULL to make the service choose a voice based on languageCode and gender.
audioEncoding	Format of the requested audio stream
speakingRate	Speaking rate/speed between 0.25 and 4.0
pitch	Speaking pitch between -20.0 and 20.0 in semitones.
volumeGainDb	Volumne gain in dB
sampleRateHertz	Sample rate for returned audio
inputType	Choose between text (the default) or SSML markup. The input text must be SSML markup if you choose ssml
effectsProfileIds	Optional. An identifier which selects 'audio effects' profiles that are applied on (post synthesized) text to speech. Effects are applied on top of each other in the order they are given

Details

Requires the Cloud Text-To-Speech API to be activated for your Google Cloud project.

Supported voices are here https://cloud.google.com/text-to-speech/docs/voices and can be imported into R via gl_talk_languages

To play the audio in code via a browser see gl_talk_player

To use Speech Synthesis Markup Language (SSML) select inputType=ssml - more details on using this to insert pauses, sounds and breaks in your audio can be found here: https://cloud.google.com/text-to-speech/docs/ssml

To use audio profiles, supply a character vector of the available audio profiles listed here: https://cloud.google.com/text-to-speech/docs/audio-profiles - the audio profiles are applied in the order given. For instance effectsProfileIds="wearable-class-device" will optimise output for smart watches, effectsProfileIds=c("wearable-class-device","telephony-class-application") will apply sound filters optimised for smart watches, then telephonic devices.

Value

The file output name you supplied as output

See Also

https://cloud.google.com/text-to-speech/docs/

Examples

```
## Not run:
library(magrittr)
gl_talk("The rain in spain falls mainly in the plain",
        output = "output.wav")
```

```
gl_talk("Testing my new audio player") %>% gl_talk_player()

# using SSML
gl_talk('<speak>The <say-as interpret-as=\"characters\">SSML</say-as>
  standard <break time=\"1s\"/>is defined by the
  <sub alias=\"World Wide Web Consortium\">W3C</sub>.</speak>',
  inputType =  "ssml")

# using effects profiles
gl_talk("This sounds great on headphones",
        effectsProfileIds = "headphone-class-device")


## End(Not run)
```

`gl_talk_languages` *Get a list of voices available for text to speech*

Description

Returns a list of voices supported for synthesis.

Usage

```
gl_talk_languages(languageCode = NULL)
```

Arguments

`languageCode`	A BCP-47 language tag. If specified, will only return voices that can be used to synthesize this languageCode

---

`gl_talk_player` *Play audio in a browser*

Description

This uses HTML5 audio tags to play audio in your browser

Usage

```
gl_talk_player(audio = "output.wav", html = "player.html")
```

Arguments

`audio`	The file location of the audio file. Must be supported by HTML5
`html`	The html file location that will be created host the audio

Details

A platform neutral way to play audio is not easy, so this uses your browser to play it instead.

Examples

```
## Not run:

gl_talk("Testing my new audio player") %>% gl_talk_player()


## End(Not run)
```

`gl_talk_shiny` *Speak in Shiny module (server)*

Description

Call via `shiny::callModule(gl_talk_shiny, "your_id")`

Usage

```
gl_talk_shiny(
  input,
  output,
  session,
  transcript,
  ...,
  autoplay = TRUE,
  controls = TRUE,
  loop = FALSE,
  keep_wav = FALSE
)
```

Arguments

`input`	shiny input
`output`	shiny output
`session`	shiny session
`transcript`	The (reactive) text to talk
`...`	Arguments passed on to `gl_talk`
	`languageCode` The language of the voice as a `BCP-47` language code
	`name` Name of the voice, see list via gl_talk_languages for supported voices. Set to `NULL` to make the service choose a voice based on `languageCode` and `gender`.
	`gender` The gender of the voice, if available

audioEncoding Format of the requested audio stream

speakingRate Speaking rate/speed between 0.25 and 4.0

pitch Speaking pitch between -20.0 and 20.0 in semitones.

volumeGainDb Volumne gain in dB

sampleRateHertz Sample rate for returned audio

inputType Choose between text (the default) or SSML markup. The input text must be SSML markup if you choose ssml

effectsProfileIds Optional. An identifier which selects 'audio effects' profiles that are applied on (post synthesized) text to speech. Effects are applied on top of each other in the order they are given

autoplay	passed to the HTML audio player - default TRUE plays on load
controls	passed to the HTML audio player - default TRUE shows controls
loop	passed to the HTML audio player - default FALSE does not loop
keep_wav	keep the generated wav files if TRUE.

---

gl_talk_shinyUI *Speak in Shiny module (ui)*

Description

Speak in Shiny module (ui)

Usage

```
gl_talk_shinyUI(id)
```

Arguments

id	The Shiny id

Details

Shiny Module for use with gl_talk_shiny.

`gl_translate` *Translate the language of text within a request*

Description

Translate character vectors via the Google Translate API

Usage

```
gl_translate(
  t_string,
  target = "en",
  format = c("text", "html"),
  source = "",
  model = c("nmt", "base")
)
```

Arguments

`t_string`	A character vector of text to detect language for
`target`	The target language
`format`	Whether the text is plain or HTML
`source`	Specify the language to translate from. Will detect it if left default
`model`	What translation model to use

Details

You can translate a vector of strings, although if too many for one call then it will be broken up into one API call per element. This is the same cost as charging is per character translated, but will take longer.

If translating HTML set the `format = "html"`. Consider removing anything not needed to be translated first, such as JavaScript and CSS scripts. See example on how to do this with `rvest`

The API limits in three ways: characters per day, characters per 100 seconds, and API requests per 100 seconds. All can be set in the API manager https://console.developers.google.com/apis/api/translate.googleapis.com/quotas

Value

A tibble of `translatedText` and `detectedSourceLanguage` and `text` of length equal to the vector of text you passed in.

See Also

https://cloud.google.com/translate/docs/reference/translate

Other translations: `gl_translate_detect()`, `gl_translate_languages()`

Examples

```
## Not run:

text <- "to administer medicine to animals is frequently a very difficult matter,
  and yet sometimes it's necessary to do so"

gl_translate(text, target = "ja")

# translate webpages using rvest to process beforehand
library(rvest)
library(googleLanguageR)

# translate webpages

# dr.dk article
my_url <- "http://bit.ly/2yhrmrH"

## in this case the content to translate is in css selector '.wcms-article-content'
read_html(my_url) %>%
  html_node(css = ".wcms-article-content") %>%
  html_text %>%
  gl_translate(format = "html")


## End(Not run)
```

`gl_translate_detect` *Detect the language of text within a request*

Description

Detect the language of text within a request

Usage

```
gl_translate_detect(string)
```

Arguments

`string`	A character vector of text to detect language for

Details

Consider using `library(cld2)` and `cld2::detect_language` instead offline, since that is free and local without needing a paid API call.

gl_translate also returns a detection of the language, so you could also wish to do it in one step via that function.

Value

A tibble of the detected languages with columns `confidence`, `isReliable`, `language`, and `text` of length equal to the vector of text you passed in.

See Also

https://cloud.google.com/translate/docs/reference/detect

Other translations: `gl_translate_languages()`, `gl_translate()`

Examples

```
## Not run:

gl_translate_detect("katten sidder på måtten")
# Detecting language: 39 characters - katten sidder på måtten...
# confidence isReliable language                    text
# 1   0.536223      FALSE       da katten sidder på måtten



## End(Not run)
```

---

gl_translate_languages

Lists languages from Google Translate API

Description

Returns a list of supported languages for translation.

Usage

```
gl_translate_languages(target = "en")
```

Arguments

`target`	If specified, language names are localized in target language

Details

Supported language codes, generally consisting of its ISO 639-1 identifier. (E.g. `'en'`, `'ja'`). In certain cases, BCP-47 codes including language + region identifiers are returned (e.g. `'zh-TW'`, `'zh-CH'`)

Value

A tibble of supported languages

See Also

https://cloud.google.com/translate/docs/reference/languages

Other translations: gl_translate_detect(), gl_translate()

Examples

```
## Not run:

# default english names of languages supported
gl_translate_languages()

# specify a language code to get other names, such as Danish
gl_translate_languages("da")


## End(Not run)
```

---

googleLanguageR *googleLanguageR*

Description

This package contains functions for analysing language through the Google Cloud Machine Learning APIs

Details

For examples and documentation see the vignettes and the website:

http://code.markedmondson.me/googleLanguageR/

See Also

https://cloud.google.com/products/machine-learning/

Index