Package 'mscsweblm4r'

May 9, 2026

Type Package

Title R Client for the Microsoft Cognitive Services Web Language Model REST API

Version 0.1.2

Maintainer Phil Ferriere <pferriere@hotmail.com>

Description R Client for the Microsoft Cognitive Services Web Language Model REST API, including Break Into Words, Calculate Conditional Probability, Calculate Joint Probability, Generate Next Words, and List Available Models. A valid 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/mscsweblm4r

BugReports http://www.github.com/philferriere/mscsweblm4r/issues

VignetteBuilder knitr

Imports methods, httr, jsonlite, pander

Suggests knitr, rmarkdown, testthat, mscstexta4r

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 2016-06-15 22:02:43

Contents

`mscsweblm4r` *R Client for the Microsoft Cognitive Services Web Language Model REST API*

Description

mscsweblm4r is a client/wrapper/interface for the Microsoft Cognitive Services (MSCS) Web Language Model (Web LM) 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 Web LM 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; and speech and language understanding.

The Web Language Model REST API provides tools for natural language processing and is documented at https://www.microsoft.com/cognitive-services/en-us/web-language-model-api/documentation. Per Microsoft's website, this API uses smoothed backoff N-gram language models (supporting Markov order up to 5) that were trained on four web-scale American English corpora collected by Bing (web page body, title, anchor and query).

The MSCS Web LM REST API supports the following lookup operations:

- Insert spaces into a string of words adjoined together without any spaces (hashtags, URLs, etc.).
- Calculate the joint probability that a sequence of words will appear together.
- Compute the conditional probability that a specific word will follow an existing sequence of words.
- Get the list of words (completions) most likely to follow a given sequence of words.
- Retrieve the list of supported language models.

mscsweblm4r Functions

The following five **mscsweblm4r** core functions are used to wrap the MSCS Web LM REST API:

- Word breaking - `weblmBreakIntoWords` function
- Joint probability - `weblmCalculateJointProbability` function
- Conditional probability - `weblmCalculateConditionalProbability` function
- Sequence completions - `weblmGenerateNextWords` function
- Models list - `weblmListAvailableModels` function

The `weblmInit` 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.

Package Loading and Configuration

After loading the **mscsweblm4r** package with the `library()` function, you must call the `weblmInit` before you can call any of the core **mscsweblm4r** functions.

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

If `MSCS_WEBLANGUAGEMODEL_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:

```
{
  "weblanguagemodelurl": "https://api.projectoxford.ai/text/weblm/v1.0/",
  "weblanguagemodelkey": "...MSCS Web Language Model API key goes here..."
}
```

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

`MSCS_WEBLANGUAGEMODEL_URL` - the URL for the Web LM REST API.

`MSCS_WEBLANGUAGEMODEL_KEY` - your personal Web LM REST API key.

S3 Object of the Class `weblm`

The five core functions of the **mscsweblm4r** package return S3 objects of the class `weblm`. Those objects expose formatted results, the REST API JSON response, and the HTTP request.

Error Handling

The MSCS Web LM 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.

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 **mscsweblm4r**'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>

`weblm` *The* `weblm` *object*

Description

The `weblm` 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>

`weblmBreakIntoWords` *Breaks a string of concatenated words into individual words*

Description

This function inserts spaces into a string of words lacking spaces, like a hashtag or part of a URL. Punctuation or exotic characters can prevent a string from being broken, so it's best to limit input strings to lower-case, alpha-numeric characters. The input string must be in ASCII format.

Internally, this function invokes the Microsoft Cognitive Services Web Language Model REST API documented at https://www.microsoft.com/cognitive-services/en-us/web-language-model-api/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

```
weblmBreakIntoWords(textToBreak, modelToUse = "body", orderOfNgram = 5L,
  maxNumOfCandidatesReturned = 5L)
```

Arguments

textToBreak	(character) Line of text to break into words. If spaces are present, they will be interpreted as hard breaks and maintained, except for leading or trailing spaces, which will be trimmed. Must be in ASCII format.
modelToUse	(character) Which language model to use, supported values: "title", "anchor", "query", or "body" (optional, default: "body")
orderOfNgram	(integer) Which order of N-gram to use, supported values: 1L, 2L, 3L, 4L, or 5L (optional, default: 5L)
maxNumOfCandidatesReturned	(integer) Maximum number of candidates to return (optional, default: 5L)

Value

An S3 object of the class weblm. The results are stored in the results dataframe inside this object. The dataframe contains the candidate breakdowns and their log(probability).

Author(s)

Phil Ferriere <pferriere@hotmail.com>

Examples

```
## Not run:
 tryCatch({

   # Break a sentence into words
   textWords <- weblmBreakIntoWords(
     textToBreak = "testforwordbreak", # ASCII only
     modelToUse = "body",              # "title"|"anchor"|"query"(default)|"body"
     orderOfNgram = 5L,                # 1L|2L|3L|4L|5L(default)
     maxNumOfCandidatesReturned = 5L   # Default: 5L
   )

   # Class and structure of textWords
   class(textWords)
   #> [1] "weblm"

   str(textWords, max.level = 1)
   #> List of 3
   #>  $ results:'data.frame':  5 obs. of  2 variables:
   #>  $ json   : chr "{"candidates":[{"words":"test for word break", __truncated__ }]}
   #>  $ request:List of 7
   #>   ..- attr(*, "class")= chr "request"
   #>  - attr(*, "class")= chr "weblm"
```

```
  # Print results
  pandoc.table(textWords$results)
  #> ---------------------------------
  #>       words          probability
  #> ------------------- -------------
  #> test for word break    -13.83
  #>
  #>  test for wordbreak    -14.63
  #>
  #>  testfor word break    -15.94
  #>
  #>  test forword break    -16.72
  #>
  #>   testfor wordbreak    -17.41
  #> ---------------------------------

 }, error = function(err) {

  # Print error
  geterrmessage()

 })

## End(Not run)
```

`weblmCalculateConditionalProbability` *Calculates the conditional probability that a word follows a sequence of words.*

Description

This function calculates the conditional probability that a particular word will follow a given sequence of words. The input string must be in ASCII format.

Internally, this function invokes the Microsoft Cognitive Services Web Language Model REST API documented at https://www.microsoft.com/cognitive-services/en-us/web-language-model-api/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

```
weblmCalculateConditionalProbability(precedingWords, continuations,
  modelToUse = "body", orderOfNgram = 5L)
```

Arguments

precedingWords	(character) Character string for which to calculate continuation probabilities. Must be in ASCII format.
continuations	(character vector) Vector of words following precedingWords for which to calculate conditional probabilities.
modelToUse	(character) Which language model to use, supported values: "title", "anchor", "query", or "body" (optional, default: "body")
orderOfNgram	(integer) Which order of N-gram to use, supported values: 1L, 2L, 3L, 4L, or 5L (optional, default: 5L)

Value

An S3 object of the class weblm. The results are stored in the results dataframe inside this object. The dataframe contains the continuation words and their log(probability).

Author(s)

Phil Ferriere <pferriere@hotmail.com>

Examples

```
## Not run:
 tryCatch({

  # Calculate conditional probability a particular word will follow a given sequence of words
   conditionalProbabilities <- weblmCalculateConditionalProbability(
     precedingWords = "hello world wide",       # ASCII only
     continuations = c("web", "range", "open"), # ASCII only
    modelToUse = "title",                     # "title"|"anchor"|"query"(default)|"body"
     orderOfNgram = 4L                          # 1L|2L|3L|4L|5L(default)
   )

   # Class and structure of conditionalProbabilities
   class(conditionalProbabilities)
   #> [1] "weblm"

   str(conditionalProbabilities, max.level = 1)
   #> List of 3
   #>  $ results:'data.frame':  3 obs. of  3 variables:
  #> $ json  : chr "{"results":[{"words":"hello world wide","word":"web", __truncated__ }]}
   #>  $ request:List of 7
   #>   ..- attr(*, "class")= chr "request"
   #>  - attr(*, "class")= chr "weblm"

   # Print results
   pandoc.table(conditionalProbabilities$results)
   #> ------------------------------------
   #>      words        word   probability
   #> ---------------- ------ -------------
   #> hello world wide   web      -0.32
```

```
  #>
  #> hello world wide range     -2.403
  #>
  #> hello world wide  open      -2.97
  #> -------------------------------------

 }, error = function(err) {

  # Print error
  geterrmessage()

 })

## End(Not run)
```

`weblmCalculateJointProbability` *Calculates the joint probability that a sequence of words will appear together.*

Description

This function calculates the joint probability that a particular sequence of words will appear together. The input string must be in ASCII format.

Internally, this function invokes the Microsoft Cognitive Services Web Language Model REST API documented at https://www.microsoft.com/cognitive-services/en-us/web-language-model-api/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

```
weblmCalculateJointProbability(inputWords, modelToUse = "body",
  orderOfNgram = 5L)
```

Arguments

`inputWords`	(character vector) Vector of character strings for which to calculate the joint probability. Must be in ASCII format.
`modelToUse`	(character) Which language model to use, supported values: "title", "anchor", "query", or "body" (optional, default: "body")
`orderOfNgram`	(integer) Which order of N-gram to use, supported values: 1L, 2L, 3L, 4L, or 5L (optional, default: 5L)

Value

An S3 object of the class weblm. The results are stored in the results dataframe inside this object. The dataframe contains the word sequences and their log(probability).

Author(s)

Phil Ferriere <pferriere@hotmail.com>

Examples

```
## Not run:
 tryCatch({

   # Calculate joint probability a particular sequence of words will appear together
   jointProbabilities <- weblmCalculateJointProbability(
     inputWords = c("where", "is", "San", "Francisco", "where is",
                    "San Francisco", "where is San Francisco"),  # ASCII only
     modelToUse = "query",                    # "title"|"anchor"|"query"(default)|"body"
     orderOfNgram = 4L                         # 1L|2L|3L|4L|5L(default)
   )

   # Class and structure of jointProbabilities
   class(jointProbabilities)
   #> [1] "weblm"

   str(jointProbabilities, max.level = 1)
   #> List of 3
   #>  $ results:'data.frame':  7 obs. of  2 variables:
  #>  $ json   : chr "{"results":[{"words":"where","probability":-3.378}, __truncated__ ]}
   #>  $ request:List of 7
   #>   ..- attr(*, "class")= chr "request"
   #>  - attr(*, "class")= chr "weblm"

   # Print results
   pandoc.table(jointProbabilities$results)
   #> ------------------------------------
   #>         words           probability
   #> ---------------------- -------------
   #>         where             -3.378
   #>
   #>           is              -2.607
   #>
   #>          san              -3.292
   #>
   #>       francisco           -4.051
   #>
   #>        where is           -3.961
   #>
   #>     san francisco         -4.086
   #>
   #> where is san francisco    -7.998
   #> ------------------------------------
```

```
}, error = function(err) {

  # Print error
  geterrmessage()

})

## End(Not run)
```

`weblmGenerateNextWords` *Returns the words most likely to follow a sequence of words.*

Description

This function returns the list of words (completions) most likely to follow a given sequence of words. The input string must be in ASCII format.

Internally, this function invokes the Microsoft Cognitive Services Web Language Model REST API documented at https://www.microsoft.com/cognitive-services/en-us/web-language-model-api/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

```
weblmGenerateNextWords(precedingWords, modelToUse = "body",
  orderOfNgram = 5L, maxNumOfCandidatesReturned = 5L)
```

Arguments

`precedingWords`	(character) Character string to retrieve completions for. Must be in ASCII format.
`modelToUse`	(character) Which language model to use, supported values: "title", "anchor", "query", or "body" (optional, default: "body")
`orderOfNgram`	(integer) Which order of N-gram to use, supported values: 1L, 2L, 3L, 4L, or 5L (optional, default: 5L)
`maxNumOfCandidatesReturned`	(integer) Maximum number of candidates to return (optional, default: 5L)

Value

An S3 object of the class `weblm`. The results are stored in the `results` dataframe inside this object. The dataframe contains the candidate words and their log(probability).

Author(s)

Phil Ferriere <pferriere@hotmail.com>

Examples

```
## Not run:
 tryCatch({

   # Generate next words
   wordCandidates <- weblmGenerateNextWords(
     precedingWords = "how are you",  # ASCII only
     modelToUse = "title",            # "title"|"anchor"|"query"(default)|"body"
     orderOfNgram = 4L,               # 1L|2L|3L|4L|5L(default)
     maxNumOfCandidatesReturned = 5L  # Default: 5L
   )

   # Class and structure of wordCandidates
   class(wordCandidates)
   #> [1] "weblm"

   str(wordCandidates, max.level = 1)
   #> List of 3
   #>  $ results:'data.frame':  5 obs. of  2 variables:
  #> $ json  : chr "{"candidates":[{"word":"doing","probability":-1.105}, __truncated__ ]}
   #>  $ request:List of 7
   #>   ..- attr(*, "class")= chr "request"
   #>  - attr(*, "class")= chr "weblm"

   # Print results
   pandoc.table(wordCandidates$results)
   #> ---------------------
   #>   word    probability
   #> ------- -------------
   #>   doing     -1.105
   #>
   #>    in       -1.239
   #>
   #> feeling     -1.249
   #>
   #>   going     -1.378
   #>
   #>   today      -1.43
   #> ---------------------

 }, error = function(err) {

   # Print error
   geterrmessage()

 })

## End(Not run)
```

`weblmInit` *Initializes the* **mscsweblm4r** *package.*

Description

This function initializes the Microsoft Cognitive Services Web Language Model 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 **mscsweblm4r** core functions, or these functions will fail.

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

If `MSCS_WEBLANGUAGEMODEL_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:

```
{
  "weblanguagemodelurl": "https://api.projectoxford.ai/text/weblm/v1.0/",
  "weblanguagemodelkey": "...MSCS Web Language Model API key goes here..."
}
```

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

`MSCS_WEBLANGUAGEMODEL_URL` - the URL for the Web LM REST API.

`MSCS_WEBLANGUAGEMODEL_KEY` - your personal Web LM REST API key.

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

Usage

```
weblmInit()
```

Author(s)

Phil Ferriere <pferriere@hotmail.com>

Examples

```
## Not run:
 weblmInit()

## End(Not run)
```

`weblmListAvailableModels` *Retrieves the list of web language models available.*

Description

This function retrieves the list of web language models currently available.

Internally, this function invokes the Microsoft Cognitive Services Web Language Model REST API documented at https://www.microsoft.com/cognitive-services/en-us/web-language-model-api/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

```
weblmListAvailableModels()
```

Value

An S3 object of the class `weblm`. The list of available language models is stored in the `results` dataframe inside this object. The dataframe includes a short description of the corpus used to build the model, the name of the model, the max N-gram order supported, and a list of Web Language Model REST API methods supported by each model.

Author(s)

Phil Ferriere <pferriere@hotmail.com>

Examples

```
## Not run:
 tryCatch({

   # Retrieve a list of supported web language models
   modelList <- weblmListAvailableModels()

   # Class and structure of modelList
   class(modelList)          # weblm
   #> [1] "weblm"

   str(modelList, max.level = 1)
   #> List of 3
   #>  $ results:'data.frame':  4 obs. of  7 variables:
  #> $ json  : chr "{"models":[{"corpus":"bing webpage title text 2013-12", __truncated__ }]}
   #>  $ request:List of 7
   #>   ..- attr(*, "class")= chr "request"
   #>  - attr(*, "class")= chr "weblm"
```

```
  # Print partial results
  pandoc.table(modelList$results[1:3])
  #> ------------------------------------------------
  #>              corpus              model   maxOrder
  #> ------------------------------ ------- ----------
  #>    bing webpage title text      title      5
  #>             2013-12
  #>
  #> bing webpage body text 2013-12   body      5
  #>
  #>  bing web query text 2013-12    query      5
  #>
  #>    bing webpage anchor text    anchor      5
  #>             2013-12
  #> ------------------------------------------------


 }, error = function(err) {

  # Print error
  geterrmessage()

 })

## End(Not run)
```

Index