

Package: elbird (via r-universe)

October 14, 2026

Title Blazing Fast Morphological Analyzer Based on Kiwi(Korean Intelligent Word Identifier)

Version 0.3.1

Author Chanyub Park [aut, cre]
(<https://orcid.org/0000-0001-6474-2570>)

Maintainer Chanyub Park <mrchypark@gmail.com>

Description This is the R wrapper package Kiwi(Korean Intelligent Word Identifier), a blazing fast speed morphological analyzer for Korean. It supports configuration of user dictionary and detection of unregistered nouns based on frequency.

License LGPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

URL <https://github.com/mrchypark/elbird/>

BugReports <https://github.com/mrchypark/elbird/issues>

SystemRequirements A valid Kiwi installation. Dynamic library is downloaded from <<https://github.com/bab2min/Kiwi/releases>> during the build step. See <<https://github.com/bab2min/Kiwi>> as well as for API documentation. A C++17 compatible compiler is required for Kiwi v0.22.2. git, curl and cmake required to build from source for kiwi.

Depends R (>= 3.5)

Imports dplyr, purrr, tibble, R6 (>= 2.4.0), vroom, matchr

LinkingTo cpp11

Suggests covr, testthat (>= 3.0.0)

Config/testthat/edition 3

Config/pak/sysreqs cmake git

Repository <https://mrchypark.r-universe.dev>

Date/Publication 2025-12-30 11:56:35 UTC
RemoteUrl https://github.com/mrchypark/elbird
RemoteRef HEAD
RemoteSha 0e5c19633a9d021de6bfc88e35fd5056ff53b51e

Contents

analyze	2
Dialect	4
get_model	4
Joiner	5
Kiwi	5
Match	11
model_exists	11
model_home	12
model_works	12
Morphset	13
Pretokenized	15
split_into_sents	18
Stopwords	18
SwTokenizer	20
Tags	21
tokenize	21
TypoSet	23
Index	25

analyze	<i>Simple version of analyze function.</i>
---------	--

Description

Simple version of analyze function.

Usage

```
analyze(  
  text,  
  top_n = 3,  
  match_option = Match$ALL,  
  stopwords = FALSE,  
  blocklist = NULL,  
  pretokenized = NULL,  
  normalize_coda = FALSE,  
  typos = NULL,  
  typo_cost_threshold = 2.5,
```

```

    open_ending = FALSE,
    allowed_dialects = Dialect$STANDARD,
    dialect_cost = 3
  )

```

Arguments

<code>text</code>	target text.
<code>top_n</code>	integer: Number of result. Default is 3.
<code>match_option</code>	Match : use Match. Default is Match\$ALL
<code>stopwords</code>	stopwords option. Default is FALSE which is use nothing. If TRUE, use embaded stopwords dictionary. If <code>char</code> : path of dictionary txt file, use file. If Stopwords class, use it. If not valid value, work same as FALSE.
<code>blocklist</code>	Morphset(optional): morpheme set to block from analysis results. Default is NULL.
<code>pretokenized</code>	Pretokenized(optional): pretokenized object for guided analysis. Default is NULL.
<code>normalize_coda</code>	bool(optional): apply coda normalization. Default is FALSE.
<code>typos</code>	bool(optional): enable typo correction. Default is NULL (keep current).
<code>typo_cost_threshold</code>	num(optional): typo correction cost threshold. Default is 2.5.
<code>open_ending</code>	bool(optional): keep sentence open after last morpheme. Default is FALSE.
<code>allowed_dialects</code>	Dialect(optional): allowed dialects for analysis. Default is Dialect\$STANDARD.
<code>dialect_cost</code>	num(optional): cost added to dialect morphemes. Default is 3.0.

Examples

```

## Not run:
analyze("Test text.")
analyze("Please use Korean.", top_n = 1)
analyze("Test text.", 1, Match$ALL_WITH_NORMALIZING)
analyze("Test text.", stopwords = FALSE)
analyze("Test text.", stopwords = TRUE)
analyze("Test text.", stopwords = "user_dict.txt")
analyze("Test text.", stopwords = Stopwords$new(TRUE))

# New features with Kiwi v0.21.0
kw <- Kiwi$new()
morphset <- kw$create_morphset()
analyze("Test text.", blocklist = morphset)

## End(Not run)

```

Dialect	<i>Dialect Options</i>
Description Dialect constants for analysis options. Usage <pre>Dialect</pre> Format An object of class <code>EnumGenerator</code> of length 11. Examples <pre>## Not run: Dialect Dialect\$STANDARD ## End(Not run)</pre>	
get_model	<i>Get kiwi language model file.</i>

Description

Get kiwi language model file.

Usage

```
get_model(size = "base", path = model_home(), clean = FALSE)
```

Arguments

<code>size</code>	"base" model. default is "base". Also "all" available.
<code>path</code>	path for model files. default is <code>model_home()</code> .
<code>clean</code>	remove previous model files before get new.

Source

<https://github.com/bab2min/Kiwi/releases>

Examples

```
## Not run:
  get_model("base")

## End(Not run)
```

Joiner	<i>Joiner Class</i>
--------	---------------------

Description

Joiner class provides methods to compose morphemes into text.

Examples

```
## Not run:
kw <- Kiwi$new()
joiner <- kw$create_joiner()
joiner$add(" ", "NNG")
joiner$get()

## End(Not run)
```

Kiwi	<i>Kiwi Class</i>
------	-------------------

Description

Kiwi class is provide method for korean mophological analyze result.

Methods**Public methods:**

- `Kiwi$print()`
- `Kiwi$new()`
- `Kiwi$add_user_word()`
- `Kiwi$add_pre_analyzed_words()`
- `Kiwi$add_rules()`
- `Kiwi$load_user_dictionarys()`
- `Kiwi$extract_words()`
- `Kiwi$analyze()`
- `Kiwi$tokenize()`
- `Kiwi$split_into_sents()`
- `Kiwi$get_tidytext_func()`

- `Kiwi$set_typo_correction()`
- `Kiwi$get_typo_correction_settings()`
- `Kiwi$create_morphset()`
- `Kiwi$create_pretokenized()`
- `Kiwi$clone()`

Method `print()`: print method for Kiwi objects

Usage:

```
Kiwi$print(x, ...)
```

Arguments:

`x` self

`...` ignored

Method `new()`: Create a kiwi instance.

Usage:

```
Kiwi$new(
  num_workers = 0,
  model_size = "base",
  integrate_allomorph = TRUE,
  load_default_dict = TRUE
)
```

Arguments:

`num_workers` int(optional): use multi-thread core number. default is 0 which means use all core.

`model_size` char(optional): kiwi model select. default is "base".

`integrate_allomorph` bool(optional): default is TRUE.

`load_default_dict` bool(optional): use default dictionary. default is TRUE.

Method `add_user_word()`: add user word with pos and score

Usage:

```
Kiwi$add_user_word(word, tag, score, orig_word = "")
```

Arguments:

`word` char(required): target word to add.

`tag` Tags(required): tag information about word.

`score` num(required): score information about word.

`orig_word` char(optional): origin word.

Method `add_pre_analyzed_words()`: TODO

Usage:

```
Kiwi$add_pre_analyzed_words(form, analyzed, score)
```

Arguments:

`form` char(required): target word to add analyzed result.

`analyzed` data.frame(required): analyzed result expected.

score num(required): score information about pre analyzed result.

Method add_rules(): TODO

Usage:

```
Kiwi$add_rules(tag, pattern, replacement, score)
```

Arguments:

tag Tags(required): target tag to add rules.

pattern char(required): regular expression.

replacement char(required): replace text.

score num(required): score information about rules.

Method load_user_dictionarys(): add user dictionary using text file.

Usage:

```
Kiwi$load_user_dictionarys(user_dict_path)
```

Arguments:

user_dict_path char(required): path of user dictionary file.

Method extract_words(): Extract Noun word candidate from texts.

Usage:

```
Kiwi$extract_words(  
  input,  
  min_cnt,  
  max_word_len,  
  min_score,  
  pos_threshold,  
  apply = FALSE  
)
```

Arguments:

input char(required): target text data

min_cnt int(required): minimum count of word in text.

max_word_len int(required): max word length.

min_score num(required): minimum score.

pos_threshold num(required): pos threshold.

apply bool(optional): apply extracted word as user word dict.

Method analyze(): Analyze text to token and tag results.

Usage:

```
Kiwi$analyze(  
  text,  
  top_n = 3,  
  match_option = Match$ALL,  
  stopwords = FALSE,  
  blacklist = NULL,  
  pretokenized = NULL  
)
```

Arguments:

`text` `char(required)`: target text.
`top_n` `int(optional)`: number of result. Default is 3.
`match_option` `match_option` **Match**: use Match. Default is Match\$ALL
`stopwords` `stopwords` option. Default is FALSE which is use nothing. If TRUE, use embaded stopwords dictionary. If `char`: path of dictionary txt file, use file. If **Stopwords** class, use it. If not valid value, work same as FALSE.
`blocklist` `Morphset(optional)`: morpheme set to block from analysis results.
`pretokenized` `Pretokenized(optional)`: pretokenized object for guided analysis.

Returns: list of result.

Method `tokenize()`: Analyze text to token and pos result just top 1.

Usage:

```
Kiwi$tokenize(
  text,
  match_option = Match$ALL,
  stopwords = FALSE,
  form = "tibble"
)
```

Arguments:

`text` `char(required)`: target text.
`match_option` `match_option` **Match**: use Match. Default is Match\$ALL
`stopwords` `stopwords` option. Default is FALSE which is use nothing. If TRUE, use embaded stopwords dictionary. If `char`: path of dictionary txt file, use file. If **Stopwords** class, use it. If not valid value, work same as FALSE.
`form` `char(optional)`: return form. default is "tibble". "list", "tidytext" is available.

Method `split_into_sents()`: Some text may not split sentence by sentence. `split_into_sents` works split sentences to sentence by sentence.

Usage:

```
Kiwi$split_into_sents(text, match_option = Match$ALL, return_tokens = FALSE)
```

Arguments:

`text` `char(required)`: target text.
`match_option` `match_option` **Match**: use Match. Default is Match\$ALL
`return_tokens` `bool(optional)`: add tokenized resault.

Method `get_tidytext_func()`: set function to tidytext `unnest_tokens`.

Usage:

```
Kiwi$get_tidytext_func(match_option = Match$ALL, stopwords = FALSE)
```

Arguments:

`match_option` `match_option` **Match**: use Match. Default is Match\$ALL
`stopwords` `stopwords` option. Default is TRUE which is to use embaded stopwords dictionary. If FALSE, use not embaded stopwords dictionary. If `char`: path of dictionary txt file, use file. If **Stopwords** class, use it. If not valid value, work same as FALSE.

Returns: function

Examples:

```
\dontrun{
  kw <- Kiwi$new()
  tidytoken <- kw$get_tidytext_func()
  tidytoken("test")
}
```

Method `set_typo_correction()`: Set typo correction settings for the Kiwi instance.

Usage:

```
Kiwi$set_typo_correction(
  enabled = TRUE,
  cost_threshold = 2.5,
  custom_typos = NULL
)
```

Arguments:

`enabled` bool(optional): enable or disable typo correction. Default is TRUE.

`cost_threshold` num(optional): cost threshold for typo correction. Default is 2.5.

`custom_typos` list(optional): list of custom typo corrections. Each element should be a list with 'orig', 'error', and optionally 'cost' fields.

Examples:

```
\dontrun{
  kw <- Kiwi$new()
  # Enable with default settings
  kw$set_typo_correction(TRUE)

  # Enable with custom rules
  custom_rules <- list(
    list(orig = " ", error = " ", cost = 1.0),
    list(orig = " ", error = " ", cost = 1.5)
  )
  kw$set_typo_correction(TRUE, cost_threshold = 3.0, custom_typos = custom_rules)

  # Disable typo correction
  kw$set_typo_correction(FALSE)
}
```

Method `get_typo_correction_settings()`: Get current typo correction settings.

Usage:

```
Kiwi$get_typo_correction_settings()
```

Returns: list with typo correction settings

Method `create_morphset()`: Create a new morpheme set for blocking specific morphemes from analysis.

Usage:

```
Kiwi$create_morphset()
```

Returns: Morphset object

Method `create_pretokenized()`: Create a new pretokenized object for guided analysis.

Usage:

```
Kiwi$create_pretokenized()
```

Returns: Pretokenized object

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
Kiwi$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
kw <- Kiwi$new()
kw$analyze("test")
kw$tokenize("test")

## End(Not run)

## -----
## Method `Kiwi$get_tidytext_func`
## -----

## Not run:
kw <- Kiwi$new()
tidytoken <- kw$get_tidytext_func()
tidytoken("test")

## End(Not run)

## -----
## Method `Kiwi$set_typo_correction`
## -----

## Not run:
kw <- Kiwi$new()
# Enable with default settings
kw$set_typo_correction(TRUE)

# Enable with custom rules
custom_rules <- list(
  list(orig = " ", error = " ", cost = 1.0),
  list(orig = " ", error = " ", cost = 1.5)
)
kw$set_typo_correction(TRUE, cost_threshold = 3.0, custom_typos = custom_rules)
```

```
# Disable typo correction
kw$set_typo_correction(FALSE)

## End(Not run)
```

Match*Analyze Match Options.*

Description

ALL option contains URL, EMAIL, HASHTAG, MENTION, SERIAL.

Usage

Match

Format

An object of class `EnumGenerator` of length 22.

Examples

```
## Not run:
Match
Match$ALL

## End(Not run)
```

model_exists*Verifies if model files exists.*

Description

Verifies if model files exists.

Usage

```
model_exists(size = "all")
```

Arguments

size	model size. default is "all" which is true that all three models must be present.
-------------	---

Value

logical model files exists or not.

Examples

```
## Not run:
  get_model("small")
  model_exists("small")

## End(Not run)
```

model_home	<i>A simple exported version of kiwi_model_path() Returns the kiwi model path.</i>
------------	--

Description

TODO explain ELBIRD_MODEL_HOME

Usage

```
model_home()
```

Value

character: file path

Examples

```
model_home()
```

model_works	<i>Verifies if models work fine.</i>
-------------	--------------------------------------

Description

Verifies if models work fine.

Usage

```
model_works(size = "all")
```

Arguments

size model size. default is "all" which is true that all three models must be present.

Value

logical model work or not.

Examples

```
## Not run:
get_model("small")
model_works("small")

## End(Not run)
```

Morphset

Morphset Class

Description

Morphset class provides methods for managing morpheme sets that can be used to block specific morphemes from analysis results.

Methods

Public methods:

- `Morphset$print()`
- `Morphset$new()`
- `Morphset$add()`
- `Morphset$add_multiple()`
- `Morphset$get_handle()`
- `Morphset$size()`
- `Morphset$get_morphemes()`
- `Morphset$clear()`
- `Morphset$clone()`

Method print(): print method for Morphset objects

Usage:

`Morphset$print(x, ...)`

Arguments:

`x` self

`...` ignored

Method new(): Create a morphset instance with a C++ handle.

Usage:

`Morphset$new(handle)`

Arguments:

`handle` C++ handle for the morphset object

Method add(): Add a morpheme to the morphset.

Usage:

`Morphset$add(form, tag)`

Arguments:

`form` `char(required)`: morpheme form to add.

`tag` `char(required)`: POS tag for the morpheme.

Returns: `logical` indicating success

Method `add_multiple()`: Add multiple morphemes to the morphset.

Usage:

`Morphset$add_multiple(forms, tags)`

Arguments:

`forms` `character vector`: morpheme forms to add.

`tags` `character vector`: POS tags for the morphemes.

Returns: `logical vector` indicating success for each morpheme

Method `get_handle()`: Get the internal C++ handle for this morphset. This is used internally by the Kiwi class.

Usage:

`Morphset$get_handle()`

Returns: C++ handle

Method `size()`: Get the number of morphemes in this morphset.

Usage:

`Morphset$size()`

Returns: `integer` number of morphemes

Method `get_morphemes()`: Get a list of all morphemes in this morphset.

Usage:

`Morphset$get_morphemes()`

Returns: `list` of morphemes with form and tag

Method `clear()`: Clear all morphemes from this morphset. Note: This creates a new morphset handle, so existing references may become invalid.

Usage:

`Morphset$clear()`

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`Morphset$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
kw <- Kiwi$new()
morphset <- kw$create_morphset()
morphset$add(" ", "NNG")
result <- kw$analyze("   ", blocklist = morphset)

## End(Not run)
```

Pretokenized

Pretokenized Class

Description

Pretokenized class provides methods for managing pretokenized objects that can guide the morphological analysis process by providing predefined token boundaries and information.

Methods

Public methods:

- `Pretokenized$print()`
- `Pretokenized$new()`
- `Pretokenized$add_span()`
- `Pretokenized$add_token_to_span()`
- `Pretokenized$add_tokens_to_span()`
- `Pretokenized$get_handle()`
- `Pretokenized$span_count()`
- `Pretokenized$token_count()`
- `Pretokenized$get_span_info()`
- `Pretokenized$get_all_spans()`
- `Pretokenized$clear()`
- `Pretokenized$clone()`

Method `print()`: print method for Pretokenized objects

Usage:

```
Pretokenized$print(x, ...)
```

Arguments:

`x` self

`...` ignored

Method `new()`: Create a pretokenized instance with a C++ handle.

Usage:

```
Pretokenized$new(handle)
```

Arguments:

`handle` C++ handle for the pretokenized object

Method `add_span()`: Add a span to the pretokenized object.

Usage:

`Pretokenized$add_span(begin, end)`

Arguments:

`begin` `integer`(required): beginning position of the span.

`end` `integer`(required): ending position of the span.

Returns: `integer` span ID for adding tokens to this span

Method `add_token_to_span()`: Add a token to a specific span.

Usage:

`Pretokenized$add_token_to_span(span_id, form, tag, begin, end)`

Arguments:

`span_id` `integer`(required): ID of the span to add token to.

`form` `char`(required): token form.

`tag` `char`(required): POS tag for the token.

`begin` `integer`(required): beginning position of the token.

`end` `integer`(required): ending position of the token.

Returns: `logical` indicating success

Method `add_tokens_to_span()`: Add multiple tokens to a span at once.

Usage:

`Pretokenized$add_tokens_to_span(span_id, forms, tags, begins, ends)`

Arguments:

`span_id` `integer`(required): ID of the span to add tokens to.

`forms` `character` vector: token forms.

`tags` `character` vector: POS tags for the tokens.

`begins` `integer` vector: beginning positions of the tokens.

`ends` `integer` vector: ending positions of the tokens.

Returns: `logical` vector indicating success for each token

Method `get_handle()`: Get the internal C++ handle for this pretokenized object. This is used internally by the Kiwi class.

Usage:

`Pretokenized$get_handle()`

Returns: C++ handle

Method `span_count()`: Get the number of spans in this pretokenized object.

Usage:

`Pretokenized$span_count()`

Returns: `integer` number of spans

Method `token_count()`: Get the total number of tokens across all spans.

Usage:

```
Pretokenized$token_count()
```

Returns: integer total number of tokens

Method `get_span_info()`: Get information about a specific span.

Usage:

```
Pretokenized$get_span_info(span_id)
```

Arguments:

`span_id` integer(required): ID of the span to get information for.

Returns: list with span information

Method `get_all_spans()`: Get information about all spans.

Usage:

```
Pretokenized$get_all_spans()
```

Returns: list of all spans

Method `clear()`: Clear all spans and tokens from this pretokenized object. Note: This creates a new pretokenized handle, so existing references may become invalid.

Usage:

```
Pretokenized$clear()
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
Pretokenized$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
kw <- Kiwi$new()
pt <- kw$create_pretokenized()
span_id <- pt$add_span(0, 10)
pt$add_token_to_span(span_id, " ", "NNG", 0, 3)
result <- kw$analyze("    ", pretokenized = pt)

## End(Not run)
```

<code>split_into_sents</code>	<i>Split Sentences</i>
-------------------------------	------------------------

Description

Some text may not split sentence by sentence. `split_into_sents` works split sentences to sentence by sentence.

Usage

```
split_into_sents(text, return_tokens = FALSE)
```

Arguments

`text` target text.
`return_tokens` add tokenized resault.

Examples

```
## Not run:
split_into_sents("text")
split_into_sents("text", return_tokens = TRUE)

## End(Not run)
```

<code>Stopwords</code>	<i>Stopwords Class</i>
------------------------	------------------------

Description

Stopwords is for filter result.

Methods

Public methods:

- `Stopwords$print()`
- `Stopwords$new()`
- `Stopwords$add()`
- `Stopwords$add_from_dict()`
- `Stopwords$remove()`
- `Stopwords$save_dict()`
- `Stopwords$get()`
- `Stopwords$clone()`

Method `print()`: print method for `Stopwords` objects

Usage:

```
Stopwords$print(x, ...)
```

Arguments:

x self

... ignored

Method new(): Create a stopwords object for filter stopwords on [analyze\(\)](#) and [tokenize\(\)](#) results.

Usage:

```
Stopwords$new(use_system_dict = TRUE)
```

Arguments:

use_system_dict *bool(optional)*: use system stopwords dictionary or not. Default is TRUE.

Method add(): add stopword one at a time.

Usage:

```
Stopwords$add(form = NA, tag = Tags$nnp)
```

Arguments:

form *char(optional)*: Form information. Default is NA.

tag *char(optional)*: Tag information. Default is "NNP". Please check [Tags](#).

Examples:

```
\dontrun{
  sw <- Stopwords$new()
  sw$add("word", "NNG")
  sw$add("word", Tags$nng)
}
```

Method add_from_dict(): add stopword from text file. text file need to form "TEXT/TAG". TEXT can remove like "/NNP". TAG required like "FORM/NNP".

Usage:

```
Stopwords$add_from_dict(path, dict_name = "user")
```

Arguments:

path *char(required)*: dictionary file path.

dict_name *char(optional)*: default is "user"

Method remove(): remove stopword one at a time.

Usage:

```
Stopwords$remove(form = NULL, tag = NULL)
```

Arguments:

form *char(optional)*: Form information. If form not set, remove tag in input.

tag *char(required)*: Tag information. Please check [Tags](#).

Method save_dict(): save current stopwords list in text file.

Usage:

Stopwords\$save_dict(path)

Arguments:

path **char**(required): file path to save stopwords list.

Method get(): return tibble of stopwords.

Usage:

Stopwords\$get()

Returns: a **tibble** for stopwords options for **analyze()** / **tokenize()** function.

Method clone(): The objects of this class are cloneable with this method.

Usage:

Stopwords\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

Examples

```
## Not run:
  Stopwords$new()

## End(Not run)

## -----
## Method `Stopwords$add`
## -----

## Not run:
  sw <- Stopwords$new()
  sw$add("word", "NNG")
  sw$add("word", Tags$nng)

## End(Not run)
```

SwTokenizer

SwTokenizer Class

Description

SwTokenizer class provides encode/decode helpers for subword tokenization.

Examples

```
## Not run:
kw <- Kiwi$new()
swt <- kw$create_swtokenizer("tokenizer.json")
swt$encode(" ")

## End(Not run)
```

Tags	<i>Tag list</i>
------	-----------------

Description

Tags contains tag list for elbird.

Usage

Tags

Format

An object of class `EnumGenerator` of length 47.

Source

<https://github.com/bab2min/Kiwi>

Examples

```
## Not run:
  Tags
  Tags$np

## End(Not run)
```

tokenize	<i>Simple version of tokenizer function.</i>
----------	--

Description

Simple version of tokenizer function.

Usage

```
tokenize(
  text,
  match_option = Match$ALL,
  stopwords = TRUE,
  blocklist = NULL,
  pretokenized = NULL,
  normalize_coda = FALSE,
  typos = NULL,
  typo_cost_threshold = 2.5,
  open_ending = FALSE,
  allowed_dialects = Dialect$STANDARD,
```

```

    dialect_cost = 3
)

tokenize_tbl(
  text,
  match_option = Match$ALL,
  stopwords = TRUE,
  blocklist = NULL,
  pretokenized = NULL,
  normalize_coda = FALSE,
  typos = NULL,
  typo_cost_threshold = 2.5,
  open_ending = FALSE,
  allowed_dialects = Dialect$STANDARD,
  dialect_cost = 3
)

tokenize_tidytext(
  text,
  match_option = Match$ALL,
  stopwords = TRUE,
  blocklist = NULL,
  pretokenized = NULL,
  normalize_coda = FALSE,
  typos = NULL,
  typo_cost_threshold = 2.5,
  open_ending = FALSE,
  allowed_dialects = Dialect$STANDARD,
  dialect_cost = 3
)

tokenize_tidy(
  text,
  match_option = Match$ALL,
  stopwords = TRUE,
  blocklist = NULL,
  pretokenized = NULL,
  normalize_coda = FALSE,
  typos = NULL,
  typo_cost_threshold = 2.5,
  open_ending = FALSE,
  allowed_dialects = Dialect$STANDARD,
  dialect_cost = 3
)

```

Arguments

text target text.

match_option [Match](#): use Match. Default is Match\$ALL

stopwords	stopwords option. Default is TRUE which is to use embaded stopwords dictionary. If FALSE, use not embaded stopwords dictionary. If char: path of dictionary txt file, use file. If Stopwords class, use it. If not valid value, work same as FALSE. Check analyze() how to use stopwords param.
blocklist	Morphset(optional) : morpheme set to block from analysis results. Default is NULL.
pretokenized	Pretokenized(optional) : pretokenized object for guided analysis. Default is NULL.
normalize_coda	bool(optional) : apply coda normalization. Default is FALSE.
typos	bool(optional) : enable typo correction. Default is NULL (keep current).
typo_cost_threshold	num(optional) : typo correction cost threshold. Default is 2.5.
open_ending	bool(optional) : keep sentence open after last morpheme. Default is FALSE.
allowed_dialects	Dialect(optional) : allowed dialects for analysis. Default is Dialect\$STANDARD.
dialect_cost	num(optional) : cost added to dialect morphemes. Default is 3.0.

Value

list type of result.

Examples

```
## Not run:
tokenize("Test text.")
tokenize("Please use Korean.", Match$ALL_WITH_NORMALIZING)

# New features with Kiwi v0.21.0
kw <- Kiwi$new()
morphset <- kw$create_morphset()
tokenize("Test text.", blocklist = morphset)

## End(Not run)
```

TypoSet

Typo Set Options

Description

Typo correction set constants.

Usage

TypoSet

Format

An object of class `EnumGenerator` of length 6.

Examples

```
## Not run:  
TypoSet  
TypoSet$BASIC TYPO_SET  
  
## End(Not run)
```

Index

- * datasets
 - Dialect, 4
 - Match, 11
 - Tags, 21
 - TypoSet, 23
- analyze, 2
- analyze(), 19, 20, 23
- Dialect, 4
- get_model, 4
- Joiner, 5
- Kiwi, 5
- Match, 3, 8, 11, 22
- model_exists, 11
- model_home, 12
- model_home(), 4
- model_works, 12
- Morphset, 13
- Pretokenized, 15
- split_into_sents, 18
- Stopwords, 3, 8, 18, 23
- SwTokenizer, 20
- Tags, 19, 21
- tibble, 20
- tokenize, 21
- tokenize(), 19, 20
- tokenize_tbl (*tokenize*), 21
- tokenize_tidy (*tokenize*), 21
- tokenize_tidytext (*tokenize*), 21
- TypoSet, 23