NLPretext Documentation

Artefact

Feb 18, 2021

TEXT PREPROCESSING FUNCTIONS:

The NLPretext library aimed to be a meta-library to be used to help you get started on handling your NLP use-case preprocessing.

Installation

Beware, this package has been tested on Python **3.6** & **3.7** & **3.8**, and will probably not be working under python **2.7** as **Python2.7** EOL is scheduled for December 2019.

To install this library you should first clone the repository:

pip install nlpretext

This library uses Spacy as tokenizer. Current models supported are *en_core_web_sm* and *fr_core_news_sm*. If not installed, run the following commands:

pip install https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-2.3.1/en_core_web_sm-2.3.1.tar.gz

pip install https://github.com/explosion/spacy-models/releases/download/fr_core_news_sm-2.3.0/fr_core_news_sm-2.3.0.tar.gz

CHAPTER

ONE

NLPRETEXT

1.1 nlpretext.preprocessor module

class nlpretext.preprocessor.**Preprocessor**

Bases: object

static build_pipeline(*operation_list: List[dict]*) → sklearn.pipeline.Pipeline

Build sklearn pipeline from a operation list

Parameters **operation_list** (*iterable*) – list of __operations of preprocessing

Returns

Return type sklearn.pipeline.Pipeline

pipe(*operation: Callable*, *args: Optional[dict] = None*)

Add an operation and its arguments to pipe in the preprocessor

Parameters

- **operation** (*callable*) – text preprocessing function
- **args** (*dict of arguments*) –

run(*text: str*) → str

Apply pipeline to text

Parameters **text** (*string*) – text to preprocess

Returns

Return type string

CHAPTER

TWO

NLPRETEXT.BASIC MODULE

nlpretext.basic.preprocess.**filter_non_latin_characters**(*text*) → str
Function that filters non latin characters of a text

Parameters **text** (*string*) –

Returns

Return type string

nlpretext.basic.preprocess.**fix_bad_unicode**(*text*, *normalization: str = 'NFC'*) → str
Fix unicode text that's "broken" using ftfy; this includes mojibake, HTML entities and other code cruft, and non-standard forms for display purposes.

Parameters

- **text** (*string*) –
- **({'NFC'** (*normalization*) – if 'NFC', combines characters and diacritics written using separate code points, e.g. converting "e" plus an acute accent modifier into "é"; unicode can be converted to NFC form without any change in its meaning! if 'NFKC', additional normalizations are applied that can change the meanings of characters, e.g. ellipsis characters will be replaced with three periods
- **'NFKC'** – if 'NFC', combines characters and diacritics written using separate code points, e.g. converting "e" plus an acute accent modifier into "é"; unicode can be converted to NFC form without any change in its meaning! if 'NFKC', additional normalizations are applied that can change the meanings of characters, e.g. ellipsis characters will be replaced with three periods
- **'NFD'** – if 'NFC', combines characters and diacritics written using separate code points, e.g. converting "e" plus an acute accent modifier into "é"; unicode can be converted to NFC form without any change in its meaning! if 'NFKC', additional normalizations are applied that can change the meanings of characters, e.g. ellipsis characters will be replaced with three periods
- **'NFKD'})** – if 'NFC', combines characters and diacritics written using separate code points, e.g. converting "e" plus an acute accent modifier into "é"; unicode can be converted to NFC form without any change in its meaning! if 'NFKC', additional normalizations are applied that can change the meanings of characters, e.g. ellipsis characters will be replaced with three periods

Returns

Return type string

nlpretext.basic.preprocess.**lower_text**(*text: str*)
Given `text` str, transform it into lowercase

Parameters **text** (*string*) –

Returns

Return type string

nlpretext.basic.preprocess.**normalize_whitespace**(*text*) → str

Given `text` str, replace one or more spacings with a single space, and one or more linebreaks with a single newline. Also strip leading/trailing whitespace. eg. " foo bar " -> "foo bar"

Parameters **text** (*string*) –

Returns

Return type string

nlpretext.basic.preprocess.**remove_accents**(*text*, *method: str = 'unicode'*) → str

Remove accents from any accented unicode characters in `text` str, either by transforming them into ascii equivalents or removing them entirely.

Parameters

- **text** (*str*) – raw text
- **method** (*{'unicode', 'ascii'}*) – if 'unicode', remove accented char for any unicode symbol with a direct ASCII equivalent; if 'ascii', remove accented char for any unicode symbol

  NB: the 'ascii' method is notably faster than 'unicode', but less good

Returns

Return type string

Raises **ValueError** – if `method` is not in {'unicode', 'ascii'}

nlpretext.basic.preprocess.**remove_eol_characters**(*text*) → str

Remove end of line (

) char.

text : str

str

nlpretext.basic.preprocess.**remove_multiple_spaces_and_strip_text**(*text*) → str

Remove multiple spaces, strip text, and remove '-', '*' characters.

Parameters **text** (*str*) – the text to be processed

Returns the text with removed multiple spaces and strip text

Return type string

nlpretext.basic.preprocess.**remove_punct**(*text*, *marks=None*) → str

Remove punctuation from `text` by replacing all instances of `marks` with whitespace.

Parameters

- **text** (*str*) – raw text
- **marks** (*str or None*) – If specified, remove only the characters in this string, e.g. `marks=',;:'` removes commas, semi-colons, and colons. Otherwise, all punctuation marks are removed.

Returns

Return type string

Note: When `marks=None`, Python's built-in `str.translate()` is used to remove punctuation; otherwise, a regular expression is used instead. The former's performance is about 5-10x faster.

`nlpretext.basic.preprocess.`**`remove_stopwords`**(*text: str*, *lang: str*, *custom_stopwords: Optional[list] = None*) → str

Given `text` str, remove classic stopwords for a given language and custom stopwords given as a list.

Parameters

- **`text`** (*`string`*) –
- **`lang`** (*`string`*) –
- **`custom_stopwords`** (*`list of strings`*) –

Returns

Return type string

`nlpretext.basic.preprocess.`**`replace_currency_symbols`**(*text*, *replace_with=None*) → str

Replace all currency symbols in `text` str with string specified by `replace_with` str.

Parameters

- **`text`** (*`str`*) – raw text
- **`replace_with`** (*`None or string`*) –

  if None (default), replace symbols with their standard 3-letter abbreviations (e.g. '$' with 'USD', '£' with 'GBP'); otherwise, pass in a string with which to replace all symbols (e.g. "*CURRENCY*")

Returns

Return type string

`nlpretext.basic.preprocess.`**`replace_emails`**(*text*, *replace_with='*EMAIL*'*) → str

Replace all emails in `text` str with `replace_with` str

Parameters

- **`text`** (*`string`*) –
- **`replace_with`** (*`string`*) – the string you want the email address to be replaced with.

Returns

Return type string

`nlpretext.basic.preprocess.`**`replace_numbers`**(*text*, *replace_with='*NUMBER*'*) → str

Replace all numbers in `text` str with `replace_with` str.

Parameters

- **`text`** (*`string`*) –
- **`replace_with`** (*`string`*) – the string you want the number to be replaced with.

Returns

Return type string

nlpretext.basic.preprocess.**replace_phone_numbers**(*text*, *country_to_detect: list*, *replace_with: str = '*PHONE*'*, *method: str = 'regex'*) → str

Replace all phone numbers in `text` str with `replace_with` str

Parameters

- **text** (*string*) –
- **replace_with** (*string*) – the string you want the phone number to be replaced with.
- **method** (*['regex', 'detection']*) – regex is faster but will omit a lot of numbers, while detection will catch every numbers, but takes a while.
- **country_to_detect** (*list*) – If a list of country code is specified, will catch every number formatted. Only when method = 'detection'.

Returns

Return type string

nlpretext.basic.preprocess.**replace_urls**(*text*, *replace_with: str = '*URL*'*) → str

Replace all URLs in `text` str with `replace_with` str.

Parameters

- **text** (*string*) –
- **replace_with** (*string*) – the string you want the URL to be replaced with.

Returns

Return type string

nlpretext.basic.preprocess.**unpack_english_contractions**(*text*) → str

Replace *English* contractions in `text` str with their unshortened forms. N.B. The "'d" and "'s" forms are ambiguous (had/would, is/has/possessive), so are left as-is. eg. "You're fired. She's nice." -> "You are fired. She's nice."

Parameters **text** (*string*) –

Returns

Return type string

CHAPTER

THREE

NLPRETEXT.SOCIAL MODULE

nlpretext.social.preprocess.**convert_emoji_to_text**(*text*, *code_delimiters=(':', ':')*) → str

Convert emoji to their CLDR Short Name, according to the unicode convention http://www.unicode.org/emoji/charts/full-emoji-list.html eg. –> :grinning_face:

Parameters

- **text** (*str*) –
- **code_delimiters** (*tuple of symbols around the emoji code.*) –
- **eg** (*(':',':') --> :grinning_face:*) –

Returns string

Return type str

nlpretext.social.preprocess.**extract_emojis**(*text*) → list

Function that extracts emojis from a text and translates them into words eg. "I take care of my skin :(" –> [":grinning_face:"]

Parameters **text** (*str*) –

Returns list of all emojis converted with their unicode conventions

Return type list

nlpretext.social.preprocess.**extract_hashtags**(*text*) → list

Function that extracts words preceded with a '#' eg. "I take care of my skin #selfcare#selfestim" –> ["skincare", "selfestim"]

Parameters **text** (*str*) –

Returns list of all hashtags

Return type list

nlpretext.social.preprocess.**extract_mentions**(*text*) → list

Function that extracts words preceded with a '@' eg. "I take care of my skin with @thisproduct" –> ["@thisproduct"]

Parameters **text** (*str*) –

Returns

Return type string

nlpretext.social.preprocess.**remove_emoji**(*text*) → str

Remove emoji from any str by stripping any unicode in the range of Emoji unicode as defined in the unicode convention: http://www.unicode.org/emoji/charts/full-emoji-list.html

Parameters **text** (*str*) –

Returns

Return type str

nlpretext.social.preprocess.**remove_hashtag**(*text*) → str
Function that removes words preceded with a '#' eg. "I take care of my skin #selfcare#selfestim" –> "I take care of my skin"

Parameters **text** (*str*) –

Returns text of a post without hashtags

Return type str

nlpretext.social.preprocess.**remove_html_tags**(*text*) → str
Function that removes words between < and >

Parameters **text** (*str*) –

Returns

Return type string

nlpretext.social.preprocess.**remove_mentions**(*text*) → str
Function that removes words preceded with a '@'

Parameters **text** (*str*) –

Returns

Return type string

CHAPTER

FOUR

NLPRETEXT.TOKEN MODULE

nlpretext.token.preprocess.**remove_smallwords**(*tokens*, *smallwords_threshold: int*) → list

Function that removes words which length is below a threshold [“hello”, “my”, “name”, “is”, “John”, “Doe”] –> [“hello”,”name”,”John”,”Doe”]

Parameters

- **text** (*list*) – list of strings
- **smallwords_threshold** (*int*) – threshold of small word

Returns

Return type list

nlpretext.token.preprocess.**remove_special_caracters_from_tokenslist**(*tokens*) → list

Remove tokens that doesn’t contains any number or letter. eg. [‘foo’,’bar’,’—’,“‘s”,’#’] -> [‘foo’,’bar’,“‘s”]

Parameters **tokens** (*list*) – list of tokens to be cleaned

Returns list of tokens without tokens that contains only special caracters

Return type list

nlpretext.token.preprocess.**remove_stopwords**(*tokens: list*, *lang: str*, *custom_stopwords: Optional[list] = None*) → str

Remove stopwords from a text. eg. ‘I like when you move your body !’ -> ‘I move body !’

Parameters

- **tokens** (*list(str)*) – list of tokens
- **lang** (*str*) – language iso code (e.g : “en”)
- **custom_stopwords** (*list(str)|None*) – list of custom stopwords to add. None by default

Returns tokens without stopwords

Return type list

Raises **ValueError** – When inputs is not a list

nlpretext.token.preprocess.**remove_tokens_with_nonletters**(*tokens*) → list

Inputs a list of tokens, outputs a list of tokens without tokens that includes numbers of special caracters. [‘foo’,’bar’,’124’,’34euros’] -> [‘foo’,’bar’]

Parameters **tokens** (*list*) – list of tokens to be cleaned

Returns list of tokens without tokens with numbers

Return type list

CHAPTER

FIVE

NLPRETEXT.AUGMENTATION MODULE

exception nlpretext.augmentation.text_augmentation.**CouldNotAugment**
Bases: ValueError

exception nlpretext.augmentation.text_augmentation.**UnavailableAugmenter**
Bases: ValueError

nlpretext.augmentation.text_augmentation.**are_entities_in_augmented_text**(*entities: list*, *augmented_text: str*) → bool

Given a list of entities, check if all the words associated to each entity are still present in augmented text.

Parameters

- **entities** (*list*) – entities associated to initial text, must be in the following format: [
  { 'entity': str, 'word': str, 'startCharIndex': int, 'endCharIndex': int
  }, {
  ...
  }
  ]
- **augmented_text** (*str*) –

Returns

Return type True if all entities are present in augmented text, False otherwise

nlpretext.augmentation.text_augmentation.**augment_text**(*text: str*, *method: str*, *stopwords: Optional[List[str]] = None*, *entities: Optional[list] = None*) → Tuple[str, list]

Given a text with or without associated entities, generate a new text by modifying some words in the initial one, modifications depend on the chosen method (substitution with synonym, addition, deletion). If entities are given as input, they will remain unchanged. If you want some words other than entities to remain unchanged, specify it within the stopwords argument.

Parameters

- **text** (*string*) –

- **method** (*{'wordnet_synonym', 'aug_sub_bert'}*) – augmenter to use ('wordnet_synonym' or 'aug_sub_bert')
- **stopwords** (*list, optional*) – list of words to freeze throughout the augmentation
- **entities** (*list, optional*) – entities associated to text if any, must be in the following format: [

  { 'entity': str, 'word': str, 'startCharIndex': int, 'endCharIndex': int

  }, {

  ...

  }

  ]

Returns

Return type Augmented text and optional augmented entities

nlpretext.augmentation.text_augmentation.**check_interval_included**(*element1: dict*, *element2: dict*) → Optional[Tuple[dict, dict]]

Comparison of two entities on start and end positions to find if they are nested

Parameters

- **element1** (*dict*) –
- **element2** (*dict*) – both of them in the following format {

  'entity': str, 'word': str, 'startCharIndex': int, 'endCharIndex': int

  }

Returns

- *If there is an entity to remove among the two returns a tuple (element to remove, element to keep)*
- *If not, returns None*

nlpretext.augmentation.text_augmentation.**clean_sentence_entities**(*text: str*, *entities: list*) → list

Paired entities check to remove nested entities, the longest entity is kept

Parameters

- **text** (*str*) – augmented text
- **entities** (*list*) – entities associated to augmented text, must be in the following format: [

  { 'entity': str, 'word': str, 'startCharIndex': int, 'endCharIndex': int

  }, {

  ...

  }

  ]

Returns

Return type Cleaned entities

nlpretext.augmentation.text_augmentation.**get_augmented_entities**(*sentence_augmented: str, entities: list*) → list

Get entities with updated positions (start and end) in augmented text

Parameters

- **sentence_augmented** (*str*) – augmented text
- **entities** (*list*) – entities associated to initial text, must be in the following format: [

  { 'entity': str, 'word': str, 'startCharIndex': int, 'endCharIndex': int

  }, {

  ...

  }

  ]

Returns

Return type Entities with updated positions related to augmented text

nlpretext.augmentation.text_augmentation.**get_augmenter**(*method: str, stopwords: Optional[List[str]] = None*) → nlpaug.augmenter.word.synonym.SynonymAug

Initialize an augmenter depending on the given method.

Parameters

- **method** (*str (supported methods: wordnet_synonym and aug_sub_bert)*) –
- **stopwords** (*list*) – list of words to freeze throughout the augmentation

Returns

Return type Initialized nlpaug augmenter

nlpretext.augmentation.text_augmentation.**process_entities_and_text**(*entities: list, text: str, augmented_text: str*)

Given a list of initial entities, verify that they have not been altered by the data augmentation operation and are still in the augmented text. :param entities: entities associated to text, must be in the following format:

[

{ 'entity': str, 'word': str, 'startCharIndex': int, 'endCharIndex': int

}, {

...

}

]

Parameters

- **text** (*str*) – initial text
- **augmented_text** (*str*) – new text resulting of data augmentation operation

Returns

Return type Augmented text and entities with their updated position in augmented text

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

n

INDEX

U