FlashText Documentation

Release 1.0

Vikash Singh

Nov 09, 2018

KeywordProcessor

This module can be used to replace keywords in sentences or extract keywords from sentences. It is based on the FlashText algorithm.

CHAPTER 1

Installation

```
$ pip install flashtext
```

CHAPTER 2

Usage

Extract keywords

```
>>> from flashtext import KeywordProcessor
>>> keyword_processor = KeywordProcessor()
>>> # keyword_processor.add_keyword(<unclean name>, <standardised name>)
>>> keyword_processor.add_keyword('Big Apple', 'New York')
>>> keyword_processor.add_keyword('Bay Area')
>>> keywords_found = keyword_processor.extract_keywords('I love Big Apple and Bay␣
↪Area.')
>>> keywords_found
>>> # ['New York', 'Bay Area']
```

Replace keywords

```
>>> keyword_processor.add_keyword('New Delhi', 'NCR region')
>>> new_sentence = keyword_processor.replace_keywords('I love Big Apple and new␣
↪delhi.')
>>> new_sentence
>>> # 'I love New York and NCR region.'
```

Case Sensitive example

```
>>> from flashtext import KeywordProcessor
>>> keyword_processor = KeywordProcessor(case_sensitive=True)
>>> keyword_processor.add_keyword('Big Apple', 'New York')
>>> keyword_processor.add_keyword('Bay Area')
>>> keywords_found = keyword_processor.extract_keywords('I love big Apple and Bay␣
↪Area.')
>>> keywords_found
>>> # ['Bay Area']
```

Span of keywords extracted

```
>>> from flashtext import KeywordProcessor
>>> keyword_processor = KeywordProcessor()
>>> keyword_processor.add_keyword('Big Apple', 'New York')
>>> keyword_processor.add_keyword('Bay Area')
>>> keywords_found = keyword_processor.extract_keywords('I love big Apple and Bay␣
↪Area.', span_info=True)
>>> keywords_found
>>> # [('New York', 7, 16), ('Bay Area', 21, 29)]
```

Get Extra information with keywords extracted

```
>>> from flashtext import KeywordProcessor
>>> kp = KeywordProcessor()
>>> kp.add_keyword('Taj Mahal', ('Monument', 'Taj Mahal'))
>>> kp.add_keyword('Delhi', ('Location', 'Delhi'))
>>> kp.extract_keywords('Taj Mahal is in Delhi.')
>>> # [('Monument', 'Taj Mahal'), ('Location', 'Delhi')]
>>> # NOTE: replace_keywords feature won't work with this.
```

No clean name for Keywords

```
>>> from flashtext import KeywordProcessor
>>> keyword_processor = KeywordProcessor()
>>> keyword_processor.add_keyword('Big Apple')
>>> keyword_processor.add_keyword('Bay Area')
>>> keywords_found = keyword_processor.extract_keywords('I love big Apple and Bay␣
↪Area.')
>>> keywords_found
>>> # ['Big Apple', 'Bay Area']
```

Add Multiple Keywords simultaneously

```
>>> from flashtext import KeywordProcessor
>>> keyword_processor = KeywordProcessor()
>>> keyword_dict = {
>>>     "java": ["java_2e", "java programing"],
>>>     "product management": ["PM", "product manager"]
>>> }
>>> # {'clean_name': ['list of unclean names']}
>>> keyword_processor.add_keywords_from_dict(keyword_dict)
>>> # Or add keywords from a list:
>>> keyword_processor.add_keywords_from_list(["java", "python"])
>>> keyword_processor.extract_keywords('I am a product manager for a java_2e␣
↪platform')
>>> # output ['product management', 'java']
```

To Remove keywords

```
>>> from flashtext import KeywordProcessor
>>> keyword_processor = KeywordProcessor()
>>> keyword_dict = {
>>>     "java": ["java_2e", "java programing"],
>>>     "product management": ["PM", "product manager"]
>>> }
>>> keyword_processor.add_keywords_from_dict(keyword_dict)
>>> print(keyword_processor.extract_keywords('I am a product manager for a java_
↪2e platform'))
```

(continues on next page)

(continued from previous page)

```
>>> # output ['product management', 'java']
>>> keyword_processor.remove_keyword('java_2e')
>>> # you can also remove keywords from a list/ dictionary
>>> keyword_processor.remove_keywords_from_dict({"product management": ["PM"]})
>>> keyword_processor.remove_keywords_from_list(["java programing"])
>>> keyword_processor.extract_keywords('I am a product manager for a java_2e␣
↪platform')
>>> # output ['product management']
```

To check Number of terms in KeywordProcessor

```
>>> from flashtext import KeywordProcessor
>>> keyword_processor = KeywordProcessor()
>>> keyword_dict = {
>>>     "java": ["java_2e", "java programing"],
>>>     "product management": ["PM", "product manager"]
>>> }
>>> keyword_processor.add_keywords_from_dict(keyword_dict)
>>> print(len(keyword_processor))
>>> # output 4
```

To check if term is present in KeywordProcessor

```
>>> from flashtext import KeywordProcessor
>>> keyword_processor = KeywordProcessor()
>>> keyword_processor.add_keyword('j2ee', 'Java')
>>> 'j2ee' in keyword_processor
>>> # output: True
>>> keyword_processor.get_keyword('j2ee')
>>> # output: Java
>>> keyword_processor['colour'] = 'color'
>>> keyword_processor['colour']
>>> # output: color
```

Get all keywords in dictionary

```
>>> from flashtext import KeywordProcessor
>>> keyword_processor = KeywordProcessor()
>>> keyword_processor.add_keyword('j2ee', 'Java')
>>> keyword_processor.add_keyword('colour', 'color')
>>> keyword_processor.get_all_keywords()
>>> # output: {'colour': 'color', 'j2ee': 'Java'}
```

For detecting Word Boundary currently any character other than this *\w [A-Za-z0-9_]* is considered a word boundary.

To set or add characters as part of word characters

```
>>> from flashtext import KeywordProcessor
>>> keyword_processor = KeywordProcessor()
>>> keyword_processor.add_keyword('Big Apple')
>>> print(keyword_processor.extract_keywords('I love Big Apple/Bay Area.'))
>>> # ['Big Apple']
>>> keyword_processor.add_non_word_boundary('/')
>>> print(keyword_processor.extract_keywords('I love Big Apple/Bay Area.'))
>>> # []
```

CHAPTER 3

API doc

3.1 API Doc

3.1.1 Import and initialize module

```
>>> from flashtext import KeywordProcessor
>>> keyword_processor = KeywordProcessor()
>>> # if match has to be case sensitive
>>> keyword_processor = KeywordProcessor(case_sensitive=True)
```

3.1.2 Add Keywords to module

```
>>> keyword_processor.add_keyword('Big Apple', 'New York')
>>> keyword_processor.add_keyword('Bay Area')
```

3.1.3 Extract keywords

```
>>> keywords_found = keyword_processor.extract_keywords('I love Big Apple and Bay␣
↪Area.')
>>> keywords_found
>>> ['New York', 'Bay Area']
```

3.1.4 Replace keywords

```
>>> keyword_processor.add_keyword('New Delhi', 'NCR region')
>>> new_sentence = keyword_processor.replace_keywords('I love Big Apple and new delhi.
↪')
```

(continues on next page)

(continued from previous page)

```
>>> new_sentence
>>> 'I love New York and NCR region.'
```

3.1.5 Add keywords from File

```
>>> # Option 1: keywords.txt content
>>> # java_2e=>java
>>> # java programing=>java
>>> # product management=>product management
>>> # product management techniques=>product management
```

```
>>> # Option 2: keywords.txt content
>>> # java
>>> # python
>>> # c++
```

```
>>> keyword_processor.add_keyword_from_file('keywords.txt')
```

3.1.6 Add keywords from dict

```
>>> keyword_dict = {
        "java": ["java_2e", "java programing"],
        "product management": ["PM", "product manager"]
    }
>>> keyword_processor.add_keywords_from_dict(keyword_dict)
```

3.1.7 Add keywords from list

```
>>> keyword_processor.add_keywords_from_list(["java", "python"]})
```

3.2 KeywordProcessor Class Doc

class flashtext.keyword.**KeywordProcessor**(*case_sensitive=False*)

Attributes:

_keyword (str): Used as key to store keywords in trie dictionary. Defaults to '_keyword_'

non_word_boundaries (set(str)): Characters that will determine if the word is continuing. Defaults to set([A-Za-z0-9_])

keyword_trie_dict (dict): Trie dict built character by character, that is used for lookup Defaults to empty dictionary

case_sensitive (boolean): if the search algorithm should be case sensitive or not. Defaults to False

Examples:

```
>>> # import module
>>> from flashtext import KeywordProcessor
>>> # Create an object of KeywordProcessor
>>> keyword_processor = KeywordProcessor()
>>> # add keywords
>>> keyword_names = ['NY', 'new-york', 'SF']
>>> clean_names = ['new york', 'new york', 'san francisco']
>>> for keyword_name, clean_name in zip(keyword_names, clean_names):
>>>     keyword_processor.add_keyword(keyword_name, clean_name)
>>> keywords_found = keyword_processor.extract_keywords('I love SF and NY.␣
↪new-york is the best.')
>>> keywords_found
>>> ['san francisco', 'new york', 'new york']
```

Note:

- loosely based on Aho-Corasick algorithm.
- Idea came from this Stack Overflow Question.

add_keyword(*keyword*, *clean_name=None*)

To add one or more keywords to the dictionary pass the keyword and the clean name it maps to.

Args:

keyword [string] keyword that you want to identify

clean_name [string] clean term for that keyword that you would want to get back in return or replace if not provided, keyword will be used as the clean name also.

Returns:

status [bool] The return value. True for success, False otherwise.

Examples:

```
>>> keyword_processor.add_keyword('Big Apple', 'New York')
>>> # This case 'Big Apple' will return 'New York'
>>> # OR
>>> keyword_processor.add_keyword('Big Apple')
>>> # This case 'Big Apple' will return 'Big Apple'
```

add_keyword_from_file(*keyword_file*, *encoding='utf-8'*)

To add keywords from a file

Args: keyword_file : path to keywords file encoding : specify the encoding of the file

Examples: keywords file format can be like:

```
>>> # Option 1: keywords.txt content
>>> # java_2e=>java
>>> # java programing=>java
>>> # product management=>product management
>>> # product management techniques=>product management
```

```
>>> # Option 2: keywords.txt content
>>> # java
>>> # python
>>> # c++
```

```
>>> keyword_processor.add_keyword_from_file('keywords.txt')
```

Raises: IOError: If *keyword_file* path is not valid

add_keywords_from_dict(*keyword_dict*)
To add keywords from a dictionary

Args: keyword_dict (dict): A dictionary with *str* key and (list *str*) as value

Examples:

```
>>> keyword_dict = {
        "java": ["java_2e", "java programing"],
        "product management": ["PM", "product manager"]
    }
>>> keyword_processor.add_keywords_from_dict(keyword_dict)
```

Raises: AttributeError: If value for a key in *keyword_dict* is not a list.

add_keywords_from_list(*keyword_list*)
To add keywords from a list

Args: keyword_list (list(str)): List of keywords to add

Examples:

```
>>> keyword_processor.add_keywords_from_list(["java", "python"]})
```

Raises: AttributeError: If *keyword_list* is not a list.

add_non_word_boundary(*character*)
add a character that will be considered as part of word.

Args:

character (char): Character that will be considered as part of word.

extract_keywords(*sentence*, *span_info=False*)
Searches in the string for all keywords present in corpus. Keywords present are added to a list *keywords_extracted* and returned.

Args: sentence (str): Line of text where we will search for keywords

Returns: keywords_extracted (list(str)): List of terms/keywords found in sentence that match our corpus

Examples:

```
>>> from flashtext import KeywordProcessor
>>> keyword_processor = KeywordProcessor()
>>> keyword_processor.add_keyword('Big Apple', 'New York')
>>> keyword_processor.add_keyword('Bay Area')
>>> keywords_found = keyword_processor.extract_keywords('I love Big Apple␣
↪and Bay Area.')
>>> keywords_found
>>> ['New York', 'Bay Area']
```

get_all_keywords(*term_so_far=''*, *current_dict=None*)
Recursively builds a dictionary of keywords present in the dictionary And the clean name mapped to those keywords.

Args:

term_so_far [string] term built so far by adding all previous characters

current_dict [dict] current recursive position in dictionary

Returns:

terms_present [dict] A map of key and value where each key is a term in the keyword_trie_dict. And value mapped to it is the clean name mapped to it.

Examples:

```
>>> keyword_processor = KeywordProcessor()
>>> keyword_processor.add_keyword('j2ee', 'Java')
>>> keyword_processor.add_keyword('Python', 'Python')
>>> keyword_processor.get_all_keywords()
>>> {'j2ee': 'Java', 'python': 'Python'}
>>> # NOTE: for case_insensitive all keys will be lowercased.
```

get_keyword(*word*)

if word is present in keyword_trie_dict return the clean name for it.

Args:

word [string] word that you want to check

Returns:

keyword [string] If word is present as it is in keyword_trie_dict then we return keyword mapped to it.

Examples:

```
>>> keyword_processor.add_keyword('Big Apple', 'New York')
>>> keyword_processor.get('Big Apple')
>>> # New York
```

remove_keyword(*keyword*)

To remove one or more keywords from the dictionary pass the keyword and the clean name it maps to.

Args:

keyword [string] keyword that you want to remove if it's present

Returns:

status [bool] The return value. True for success, False otherwise.

Examples:

```
>>> keyword_processor.add_keyword('Big Apple')
>>> keyword_processor.remove_keyword('Big Apple')
>>> # Returns True
>>> # This case 'Big Apple' will no longer be a recognized keyword
>>> keyword_processor.remove_keyword('Big Apple')
>>> # Returns False
```

remove_keywords_from_dict(*keyword_dict*)

To remove keywords from a dictionary

Args: keyword_dict (dict): A dictionary with *str* key and (list *str*) as value

Examples:

```
>>> keyword_dict = {
        "java": ["java_2e", "java programing"],
        "product management": ["PM", "product manager"]
    }
>>> keyword_processor.remove_keywords_from_dict(keyword_dict)
```

Raises: AttributeError: If value for a key in *keyword_dict* is not a list.

remove_keywords_from_list(*keyword_list*)

To remove keywords present in list

Args: keyword_list (list(str)): List of keywords to remove

Examples:

```
>>> keyword_processor.remove_keywords_from_list(["java", "python"]})
```

Raises: AttributeError: If *keyword_list* is not a list.

replace_keywords(*sentence*)

Searches in the string for all keywords present in corpus. Keywords present are replaced by the clean name and a new string is returned.

Args: sentence (str): Line of text where we will replace keywords

Returns: new_sentence (str): Line of text with replaced keywords

Examples:

```
>>> from flashtext import KeywordProcessor
>>> keyword_processor = KeywordProcessor()
>>> keyword_processor.add_keyword('Big Apple', 'New York')
>>> keyword_processor.add_keyword('Bay Area')
>>> new_sentence = keyword_processor.replace_keywords('I love Big Apple␣
↪and bay area.')
>>> new_sentence
>>> 'I love New York and Bay Area.'
```

set_non_word_boundaries(*non_word_boundaries*)

set of characters that will be considered as part of word.

Args:

non_word_boundaries (set(str)): Set of characters that will be considered as part of word.

CHAPTER 4

Test

```
$ git clone https://github.com/vi3k6i5/flashtext
$ cd flashtext
$ pip install pytest
$ python setup.py test
```

CHAPTER 5

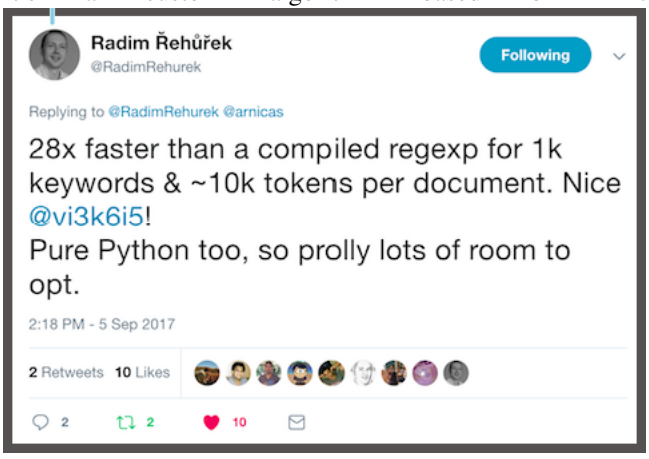
Build Docs

```
$ git clone https://github.com/vi3k6i5/flashtext
$ cd flashtext/docs
$ pip install sphinx
$ make html
$ # open _build/html/index.html in browser to view it locally
```

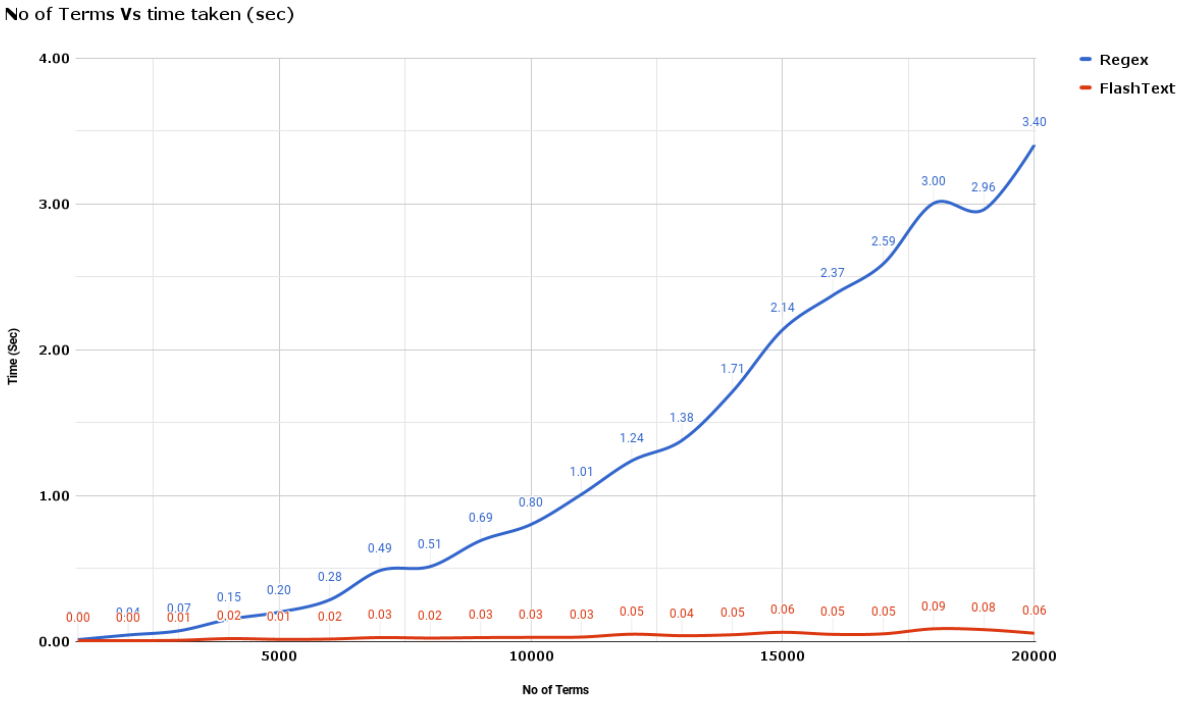
CHAPTER 6

Why not Regex?

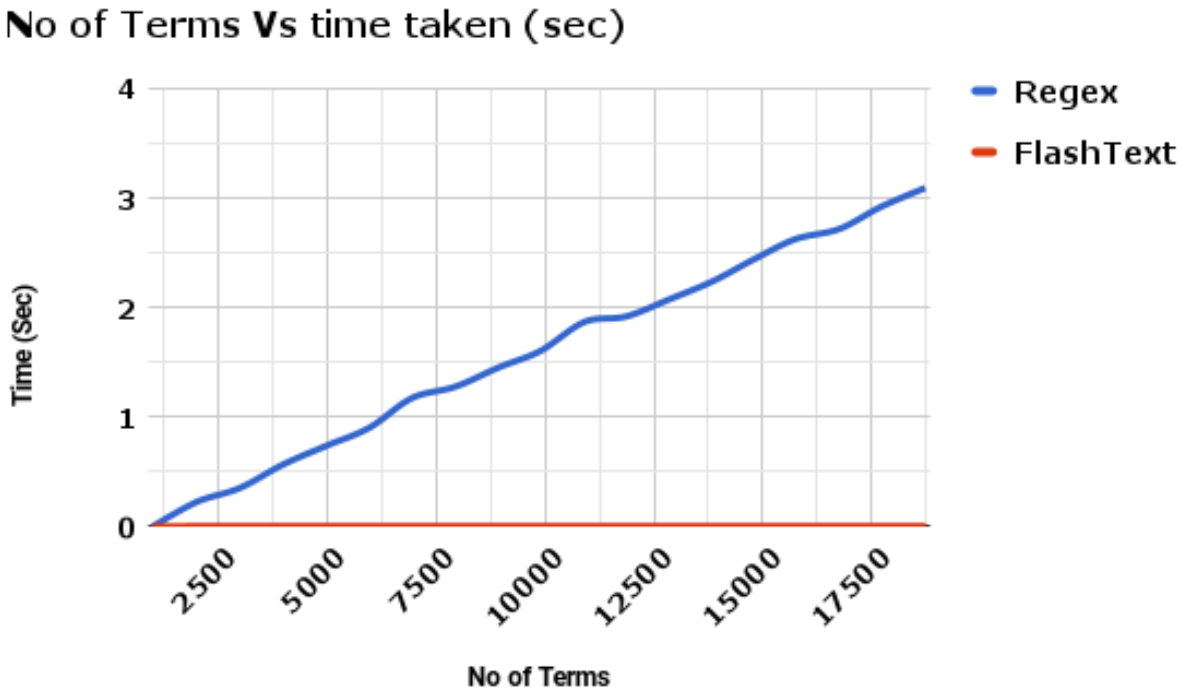
It's a custom algorithm based on Aho-Corasick algorithm and Trie Dictionary.



Time taken by FlashText to find terms in comparison to Regex.



Time taken by FlashText to replace terms in comparison to Regex.



Link to code for benchmarking the Find Feature and Replace Feature.

The idea for this library came from the following StackOverflow question.

CHAPTER 7

Citation

The original paper published on FlashText algorithm.

```
@ARTICLE{2017arXiv171100046S,
   author = {{Singh}, V.},
    title = "{Replace or Retrieve Keywords In Documents at Scale}",
  journal = {ArXiv e-prints},
archivePrefix = "arXiv",
   eprint = {1711.00046},
 primaryClass = "cs.DS",
 keywords = {Computer Science - Data Structures and Algorithms},
     year = 2017,
    month = oct,
   adsurl = {http://adsabs.harvard.edu/abs/2017arXiv171100046S},
  adsnote = {Provided by the SAO/NASA Astrophysics Data System}
}
```

The article published on Medium freeCodeCamp.

CHAPTER 8

Contribute

- Issue Tracker: https://github.com/vi3k6i5/flashtext/issues
- Source Code: https://github.com/vi3k6i5/flashtext/

CHAPTER 9

License

The project is licensed under the MIT license.

Python Module Index

f

Index