pyspellchecker Documentation

Release 0.5.2

Tyler Barrus

Sep 12, 2019

Contents

Pure Python Spell Checking based on Peter Norvig's blog post on setting up a simple spell checking algorithm.

It uses a Levenshtein Distance algorithm to find permutations within an edit distance of 2 from the original word. It then compares all permutations (insertions, deletions, replacements, and transpositions) to known words in a word frequency list. Those words that are found more often in the frequency list are **more likely** the correct results.

`pyspellchecker` supports multiple languages including English, Spanish, German, French, and Portuguese. Dictionaries were generated using the WordFrequency project on GitHub.

`pyspellchecker` supports **Python 3** and Python 2.7 but, as always, Python 3 is the preferred version!

`pyspellchecker` allows for the setting of the Levenshtein Distance to check. For longer words, it is highly recommended to use a distance of 1 and not the default 2. See the quickstart to find how one can change the distance parameter.

CHAPTER 1

Installation

The easiest method to install is using pip:

```
pip install pyspellchecker
```

To install from source:

```
git clone https://github.com/barrust/pyspellchecker.git
cd pyspellchecker
python setup.py install
```

As always, I highly recommend using the Pipenv package to help manage dependencies!

Quickstart

After installation, using `pyspellchecker` should be fairly straight forward:

```python
from spellchecker import SpellChecker

spell = SpellChecker()

# find those words that may be misspelled
misspelled = spell.unknown(['something', 'is', 'hapenning', 'here'])

for word in misspelled:
    # Get the one `most likely` answer
    print(spell.correction(word))

    # Get a list of `likely` options
    print(spell.candidates(word))
```

If the Word Frequency list is not to your liking, you can add additional text to generate a more appropriate list for your use case.

```python
from spellchecker import SpellChecker

spell = SpellChecker()  # loads default word frequency list
spell.word_frequency.load_text_file('./my_free_text_doc.txt')

# if I just want to make sure some words are not flagged as misspelled
spell.word_frequency.load_words(['microsoft', 'apple', 'google'])
spell.known(['microsoft', 'google'])  # will return both now!
```

If the words that you wish to check are long, it is recommended to reduce the *distance* to 1. This can be accomplished either when initializing the spell check class or after the fact.

```python
from spellchecker import SpellChecker

spell = SpellChecker(distance=1)  # set at initialization
```

```python
# do some work on longer words

spell.distance = 2  # set the distance parameter back to the default
```

Additional Methods

On-line documentation is available; below contains the cliff-notes version of some of the available functions:

`correction(word)`: Returns the most probable result for the misspelled word

`candidates(word)`: Returns a set of possible candidates for the misspelled word

`known([words])`: Returns those words that are in the word frequency list

`unknown([words])`: Returns those words that are not in the frequency list

`word_probability(word)`: The frequency of the given word out of all words in the frequency list

3.1 The following are less likely to be needed by the user but are available:

`edit_distance_1(word)`: Returns a set of all strings at a Levenshtein Distance of one based on the alphabet of the selected language

`edit_distance_2(word)`: Returns a set of all strings at a Levenshtein Distance of two based on the alphabet of the selected language

Credits

- Peter Norvig blog post on setting up a simple spell checking algorithm
- hermetdave's WordFrequency project for providing the basis for Non-English dictionaries

Table of contents

5.1 Quickstart

`pyspellchecker` is designed to be easy to use to get basic spell checking.

5.1.1 Installation

The best experience is likely to use `pip`:

```
pip install pyspellchecker
```

If you are using virtual environments, it is recommended to use `pipenv` to combine pip and virtual environments:

```
pipenv install pyspellchecker
```

Read more about Pipenv

5.1.2 Basic Usage

Setting up the spell checker requires importing and initializing the instance.

```python
from spellchecker import SpellChecker

spell = SpellChecker()
```

There are several methods to determine if a word is in the word frequency list:

```python
from spellchecker import SpellChecker

spell = SpellChecker()
spell['morning']  # True
```

```python
'morning' in spell  # True

# find those words from a list of words that are found in the dictionary
spell.known(['morning', 'hapenning'])  # {'morning'}

# find those words from a list of words that are not found in the dictionary
spell.unknown(['morning', 'hapenning'])  # {'hapenning'}
```

Once a word is identified as misspelled, you can find the likeliest replacement:

```python
from spellchecker import SpellChecker

spell = SpellChecker()

misspelled = spell.unknown(['morning', 'hapenning'])  # {'hapenning'}
for word in misspelled:
    spell.correction(word)  # 'happening'
```

```python
from spellchecker import SpellChecker

spell = SpellChecker(distance=1)  # set the Levenshtein Distance parameter

# do additional work

# now for shorter words, we can revert to Levenshtein Distance of 2!
spell.distance = 2
```

Or if the word identified as the likeliest is not correct, a list of candidates can also be pulled:

```python
from spellchecker import SpellChecker

spell = SpellChecker()

misspelled = spell.unknown(['morning', 'hapenning'])  # {'hapenning'}
for word in misspelled:
    spell.correction(word)  # {'penning', 'happening', 'henning'}
```

5.1.3 Changing Language

To set the language of the dictionary to load, one must set the language parameter on initialization.

```python
from spellchecker import SpellChecker

spell = SpellChecker(language='es')  # Spanish dictionary
print(spell['mañana'])
```

5.1.4 Adding and Removing Terms from a Dictionary

There are several ways to add additional terms to your word frequency dictionary including by filepath, string of text, or by a list of words.

To load a pre-defined dictionary file (either as a json file or a gzipped json file):

```
from spellchecker import SpellChecker

spell = SpellChecker()
spell.word_frequency.load_dictionary('./path-to-my-word-frequency.json')
```

To load a text document that will be parsed into individual words and each word added to the frequency list:

```
from spellchecker import SpellChecker

spell = SpellChecker()
spell.word_frequency.load_text_file('./path-to-my-text-doc.txt')
```

To load plain text from input or another source:

```
from spellchecker import SpellChecker

spell = SpellChecker()
spell.word_frequency.load_text('Text to be parsed and added to the system')
```

Or update using a list of words:

```
from spellchecker import SpellChecker

spell = SpellChecker()
spell.word_frequency.load_words(['Text', 'to', 'be','added', 'to', 'the', 'system'])
```

Or add a single word:

```
from spellchecker import SpellChecker

spell = SpellChecker()
spell.word_frequency.add('Text')
```

Removing words is as simple as adding words:

```
from spellchecker import SpellChecker

spell = SpellChecker()
spell.word_frequency.remove_words(['Text', 'to', 'be','removed', 'from', 'the',
→'system'])

# or remove a single word
spell.word_frequency.remove('meh')
```

5.1.5 How to Build a New Dictionary

Building a custom or new language dictionary is relatively straight forward. To begin, you will need to have either a word frequency list or text files that represent the usage of the terms. Since *pyspellchecker* uses word frequency, it is better to have the most common words have higher frequencies!

Once you have the corpus, code similar to the following should build out the dictionary:

```
from spellchecker import SpellChecker

# turn off loading a built language dictionary, case sensitive on (if desired)
spell = SpellChecker(language=None, case_sensitive=True)
```

```python
# if you have a dictionary...
spell.word_frequency.load_dictionary('./path-to-my-json-dictionary.json')

# or... if you have text
spell.word_frequency.load_text_file('./path-to-my-text-doc.txt')

# export it out for later use!
spell.export('my_custom_dictionary.gz', gzipped=True)
```

5.1.6 A quick, command line spell checking program

Setting up a quick and easy command line program using pyspellchecker is straight forward:

```python
from spellchecker import SpellChecker

# could add command line arguments to set the parameters of the spell
# check class; setup what type of information to present back, etc.
spell = SpellChecker()

print("To exit, hit return without input!")
while True:
    word = input('Input a word to spell check: ')
    if word == '':  # not sure, but need a way to kill the program...
        break
    word = word.lower()
    if word in spell:
        print("'{}' is spelled correctly!".format(word))
    else:
        cor = spell.correction(word)
        print("The best spelling for '{}' is '{}'".format(word, cor))

        print("If that is not enough; here are all possible candidate words:")
        print(spell.candidates(word))
```

5.2 pyspellchecker API

Here you can find the full developer API for the pyspellchecker project. pyspellchecker provides a library for determining if a word is misspelled and what the likely correct spelling would be based on word frequency.

5.2.1 SpellChecker

class spellchecker.**SpellChecker**(*language=u'en'*, *local_dictionary=None*, *distance=2*, *tokenizer=None*, *case_sensitive=False*)

The SpellChecker class encapsulates the basics needed to accomplish a simple spell checking algorithm. It is based on the work by Peter Norvig (https://norvig.com/spell-correct.html)

> **Parameters**
>
> - **language** (*str*) – The language of the dictionary to load or None for no dictionary. Supported languages are *en*, *es*, *de*, fr' and *pt*. Defaults to *en*

- **local_dictionary** (*str*) – The path to a locally stored word frequency dictionary; if provided, no language will be loaded

- **distance** (*int*) – The edit distance to use. Defaults to 2.

- **case_sensitive** (*bool*) – Flag to use a case sensitive dictionary or not, only available when not using a language dictionary.

Note: Using a case sensitive dictionary can be slow to correct words.

---

candidates(*word*)

Generate possible spelling corrections for the provided word up to an edit distance of two, if and only when needed

> **Parameters word** (*str*) – The word for which to calculate candidate spellings
>
> **Returns** The set of words that are possible candidates
>
> **Return type** set

correction(*word*)

The most probable correct spelling for the word

> **Parameters word** (*str*) – The word to correct
>
> **Returns** The most likely candidate
>
> **Return type** str

distance

The maximum edit distance to calculate

Note: Valid values are 1 or 2; if an invalid value is passed, defaults to 2

> **Type** int

edit_distance_1(*word*)

Compute all strings that are one edit away from *word* using only the letters in the corpus

> **Parameters word** (*str*) – The word for which to calculate the edit distance
>
> **Returns** The set of strings that are edit distance one from the provided word
>
> **Return type** set

edit_distance_2(*word*)

Compute all strings that are two edits away from *word* using only the letters in the corpus

> **Parameters word** (*str*) – The word for which to calculate the edit distance
>
> **Returns** The set of strings that are edit distance two from the provided word
>
> **Return type** set

export(*filepath*, *encoding=u'utf-8'*, *gzipped=True*)

Export the word frequency list for import in the future

> **Parameters**
>
> - **filepath** (*str*) – The filepath to the exported dictionary
>
> - **encoding** (*str*) – The encoding of the resulting output

---

> - **gzipped** (*bool*) – Whether to gzip the dictionary or not

known(*words*)
>   The subset of *words* that appear in the dictionary of words

>   > **Parameters words** (*list*) – List of words to determine which are in the corpus

>   > **Returns** The set of those words from the input that are in the corpus

>   > **Return type** set

split_words(*text*)
>   Split text into individual *words* using either a simple whitespace regex or the passed in tokenizer

>   > **Parameters text** (*str*) – The text to split into individual words

>   > **Returns** A listing of all words in the provided text

>   > **Return type** list(str)

unknown(*words*)
>   The subset of *words* that do not appear in the dictionary

>   > **Parameters words** (*list*) – List of words to determine which are not in the corpus

>   > **Returns** The set of those words from the input that are not in the corpus

>   > **Return type** set

word_frequency
>   An encapsulation of the word frequency *dictionary*

>   ---

>   **Note:** Not settable

>   ---

>   > **Type** *WordFrequency*

word_probability(*word*, *total_words=None*)
>   Calculate the probability of the *word* being the desired, correct word

>   > **Parameters**

>   >   - **word** (*str*) – The word for which the word probability is calculated
>   >   - **total_words** (*int*) – The total number of words to use in the calculation; use the default for using the whole word frequency

>   > **Returns** The probability that the word is the correct word

>   > **Return type** float

5.2.2 WordFrequency

class spellchecker.**WordFrequency**(*tokenizer=None*, *case_sensitive=False*)
>   Store the *dictionary* as a word frequency list while allowing for different methods to load the data and update over time

>   **add**(*word*)
>   >   Add a word to the word frequency list

>   >   > **Parameters word** (*str*) – The word to add

dictionary

A counting dictionary of all words in the corpus and the number of times each has been seen

---

Note: Not settable

> **Type** Counter

items()

Iterator over the words in the dictionary

> **Yields** *str* – The next word in the dictionary int: The number of instances in the dictionary

---

Note: This is the same as *dict.items()*

---

keys()

Iterator over the key of the dictionary

> **Yields** *str* – The next key in the dictionary

Note: This is the same as *spellchecker.words()*

---

letters

The listing of all letters found within the corpus

Note: Not settable

> **Type** str

load_dictionary(*filename*, *encoding=u'utf-8'*)

Load in a pre-built word frequency list

> **Parameters**
>
> - **filename** (`str`) – The filepath to the json (optionally gzipped) file to be loaded
> - **encoding** (`str`) – The encoding of the dictionary

load_text(*text*, *tokenizer=None*)

Load text from which to generate a word frequency list

> **Parameters**
>
> - **text** (`str`) – The text to be loaded
> - **tokenizer** (`function`) – The function to use to tokenize a string

load_text_file(*filename*, *encoding=u'utf-8'*, *tokenizer=None*)

Load in a text file from which to generate a word frequency list

> **Parameters**
>
> - **filename** (`str`) – The filepath to the text file to be loaded
> - **encoding** (`str`) – The encoding of the text file
> - **tokenizer** (`function`) – The function to use to tokenize a string

---

load_words(*words*)
> Load a list of words from which to generate a word frequency list
>
> > **Parameters words** (`list`) – The list of words to be loaded

pop(*key*, *default=None*)
> Remove the key and return the associated value or default if not found
>
> > **Parameters**
> >
> > - **key** (`str`) – The key to remove
> >
> > - **default** (`obj`) – The value to return if key is not present

remove(*word*)
> Remove a word from the word frequency list
>
> > **Parameters word** (`str`) – The word to remove

remove_by_threshold(*threshold=5*)
> Remove all words at, or below, the provided threshold
>
> > **Parameters threshold** (`int`) – The threshold at which a word is to be removed

remove_words(*words*)
> Remove a list of words from the word frequency list
>
> > **Parameters words** (`list`) – The list of words to remove

tokenize(*text*)
> Tokenize the provided string object into individual words
>
> > **Parameters text** (`str`) – The string object to tokenize
> >
> > **Yields** *str* – The next *word* in the tokenized string

> ---
> **Note:** This is the same as the *spellchecker.split_words()*
> ---

total_words
> The sum of all word occurances in the word frequency dictionary

> ---
> **Note:** Not settable
> ---

> > **Type** int

unique_words
> The total number of unique words in the word frequency list

> ---
> **Note:** Not settable
> ---

> > **Type** int

words()
> Iterator over the words in the dictionary
>
> > **Yields** *str* – The next word in the dictionary

Note: This is the same as *spellchecker.keys()*

CHAPTER 6

Additional Information

- genindex
- modindex
- search

Index