SymSpellCppPy

Release latest

Arjun Variar & Mohit Tare

May 28, 2023

MODULES:

This library is a high-speed Python port of SymSpell v6.5, developed in C++ utilizing pybind11.

CHAPTER

ONE

INTRODUCTION

SymSpellCppPy is an optimized adaptation of SymSpell, specifically designed for Python, re-engineered in C++ and interfaced using pybind11. This implementation offers significantly enhanced speed compared to its counterparts. When compared with symspellpy, a purely Python-based SymSpell port, SymSpellCppPy is generally 3x-40x faster, offering equivalent functionalities.

Detailed documentation on SymSpell's usage and functionalities is available on the original GitHub repository: https://github.com/wolfgarbe/SymSpell

For performance benchmarks, please refer to the library homepage on GitHub: https://github.com/viig99/SymSpellCppPy

CHAPTER

TWO

INSTALLATION

SymSpellCppPy is available on PyPI and can be installed using pip:

pip install –upgrade SymSpellCppPy

2.1 Examples

This document contains examples of usage for the SymSpellCppPy library. This library is used for dictionary loading, spelling correction, and error fixing.

2.1.1 Loading the dictionary

```
import SymSpellCppPy
symSpell = SymSpellCppPy.SymSpell()
symSpell.load_dictionary(corpus="resources/frequency_dictionary_en_82_765.txt", term_
↪index=0, count_index=1, separator=" ")
```

2.1.2 Checking dictionary properties

The *SymSpell* class provides methods to inspect the loaded dictionary:

- To check the number of words in the dictionary, use the *word_count()* method:

```
print(symSpell.word_count())  # Outputs: 82781
```

- To find the length of the longest word in the dictionary, use the *max_length()* method:

```
print(symSpell.max_length())  # Outputs: 28
```

- To count the number of unique delete combinations formed, use the *entry_count()* method:

```
print(symSpell.entry_count())  # Outputs: 661047
```

2.1.3 Spelling correction

The *lookup* method allows you to find the correct spelling for a term from the dictionary:

- To find the closest spelling, use *SymSpellCppPy.Verbosity.CLOSEST*:

```
terms = symSpell.lookup("tke", SymSpellCppPy.Verbosity.CLOSEST)
print(terms[0].term)  # Outputs: "take"
```

- You can also specify a *max_edit_distance* to limit the search to terms within a certain edit distance:

```
terms = symSpell.lookup("extrine", SymSpellCppPy.Verbosity.CLOSEST, max_edit_distance=2)
print(terms[0].term)  # Outputs: "extreme"

terms = symSpell.lookup("extrine", SymSpellCppPy.Verbosity.CLOSEST, max_edit_distance=1)
print(terms)  # Outputs: []
```

2.1.4 Error fixing

SymSpellCppPy also includes features to fix compound errors and word segmentation issues in sentences:

- To fix compound errors in a sentence, use the *lookup_compound* method:

```
terms = symSpell.lookup_compound("whereis th elove hehad dated forImuch of thepast who␣
↪couqdn'tread in sixthgrade and ins pired him")
print(terms[0].term)
# Outputs: "whereas to love head dated for much of theist who couldn't read in sixth␣
↪grade and inspired him"
```

- To correct word segmentation issues in a sentence, use the *word_segmentation* method:

```
segmented_info = symSpell.word_segmentation("thequickbrownfoxjumpsoverthelazydog")
print(segmented_info.segmented_string)
# Outputs: "the quick brown fox jumps over the lazy dog"

segmented_info = symSpell.word_segmentation("thequickbrownfoxjumpsoverthelazydog")
print(segmented_info.corrected_string)
# Outputs: "they quick brown fox jumps over therapy dog"
```

2.1.5 Saving and Loading SymSpell object

To save the internal representation of a loaded *SymSpell* for fast reuse next time, use the *save_pickle* method. Do not use pickle natively:

```
symSpell.save_pickle("symspell_binary.bin")
```

To load the internal representation of a loaded *SymSpell* from a saved binary, use the *load_pickle* method:

```
anotherSymSpell = SymSpellCppPy.SymSpell()
anotherSymSpell.load_pickle("symspell_binary.bin")
terms = anotherSymSpell.lookup("tke", SymSpellCppPy.Verbosity.CLOSEST)
print(terms[0].term)
```

2.1.6 Bigram and Trigram Suggestions

The SymSpellCppPy library also supports generating bigram and trigram suggestions:

```
# To generate bigram suggestions, use the `lookup_bigram` method:
terms = symSpell.lookup_bigram("in te dh", SymSpellCppPy.Verbosity.CLOSEST)
print(terms[0].term)  # Outputs: "in the dark"

# To generate trigram suggestions, use the `lookup_trigram` method:
terms = symSpell.lookup_trigram("an plesant day", SymSpellCppPy.Verbosity.CLOSEST)
print(terms[0].term)  # Outputs: "a pleasant day"
```

2.1.7 Top N suggestions

You can also request the top N suggestions for a given word:

```
# To get the top 5 closest terms to a given word, use the `TOP` verbosity:
terms = symSpell.lookup("huse", SymSpellCppPy.Verbosity.TOP, max_edit_distance=2,␣
↪include_unknown=True)
for term in terms[:5]:
    print(term.term)
# Outputs: "house", "use", "hue", "hues", "hose"
```

2.1.8 Ignoring case and digits

By default, SymSpellCppPy is case-sensitive and considers digits as valid characters. However, you can modify this behavior:

```
# To ignore case when checking a term, use the `ignore_case` parameter:
terms = symSpell.lookup("THe", SymSpellCppPy.Verbosity.CLOSEST, ignore_case=True)
print(terms[0].term)  # Outputs: "the"

# To ignore digits when checking a term, use the `ignore_digit` parameter:
terms = symSpell.lookup("3rd", SymSpellCppPy.Verbosity.CLOSEST, ignore_digit=True)
print(terms[0].term)  # Outputs: "red"
```

2.1.9 Ignoring words with numbers

You may also choose to ignore words containing numbers:

```
# To ignore words with numbers when checking a term, use the `ignore_word_with_number`␣
↪parameter:
terms = symSpell.lookup("l33t", SymSpellCppPy.Verbosity.CLOSEST, ignore_word_with_
↪number=True)
print(terms[0].term)  # Outputs: "let"
```

2.2 SymSpellCppPy

2.2.1 SymSpellCppPy: Pybind11 binding for SymSpellPy

class SymSpellCppPy.**Info**

Bases: pybind11_object

property corrected_string

Read-only property to get the word segmented and spelling corrected string.

property distance_sum

Read-only property to get the edit distance sum between input string and corrected string.

get_corrected(*self:* SymSpellCppPy.Info) → str

Get the word segmented and spelling corrected string.

get_distance(*self:* SymSpellCppPy.Info) → int

Get the edit distance sum between input string and corrected string.

get_probability(*self:* SymSpellCppPy.Info) → float

Get the sum of word occurrence probabilities in log scale. This is a measure of how common and probable the corrected segmentation is.

get_segmented(*self:* SymSpellCppPy.Info) → str

Get the word segmented string.

property log_prob_sum

Read-only property to get the sum of word occurrence probabilities in log scale. This is a measure of how common and probable the corrected segmentation is.

property segmented_string

Read-only property to get the word segmented string.

set(*self:* SymSpellCppPy.Info, *segmented_string: str*, *corrected_string: str*, *distance_sum: int*, *log_prob_sum: float*) → None

Set the properties of Info object.

Parameters

- **segmented_string** – Word segmented string.
- **corrected_string** – Word segmented and spelling corrected string.
- **distance_sum** – Edit distance sum between input string and corrected string.
- **log_prob_sum** – Sum of word occurrence probabilities in log scale (a measure of how common and probable the corrected segmentation is).

class SymSpellCppPy.**SuggestItem**

Bases: pybind11_object

SuggestItem is a class that contains a suggested correct spelling for a misspelled word.

property count

Gets or sets the frequency of the suggestion in the dictionary (a measure of how common the word is).

property distance

Gets or sets the edit distance between the searched for word and the suggestion.

property term

Gets or sets the suggested correctly spelled word.

class SymSpellCppPy.**SymSpell**

Bases: pybind11_object

SymSpell is a class that provides fast and accurate spelling correction using Symmetric Delete spelling correction algorithm.

count_threshold(*self:* SymSpellCppPy.SymSpell) → int

Retrieves the frequency threshold to be considered as a valid word for spelling correction.

create_dictionary(*self:* SymSpellCppPy.SymSpell, *corpus: str*) → bool

Load multiple dictionary words from a file containing plain text.

create_dictionary_entry(*self:* SymSpellCppPy.SymSpell, *key: str*, *count: int*) → bool

Create or update an entry in the dictionary.

delete_dictionary_entry(*self:* SymSpellCppPy.SymSpell, *key: str*) → bool

Deletes a word from the dictionary and updates internal representation accordingly.

entry_count(*self:* SymSpellCppPy.SymSpell) → int

Retrieves the total number of delete words formed in the dictionary.

load_bigram_dictionary(*self:* SymSpellCppPy.SymSpell, *corpus: str*, *term_index: int*, *count_index: int*, *separator: str* = ' ') → bool

Load multiple dictionary entries from a file of word/frequency count pairs.

load_dictionary(*self:* SymSpellCppPy.SymSpell, *corpus: str*, *term_index: int*, *count_index: int*, *separator: str* = ' ') → bool

Load multiple dictionary entries from a file of word/frequency count pairs.

load_pickle(*self:* SymSpellCppPy.SymSpell, *filepath: str*) → None

Load internal representation from file

load_pickle_bytes(*self:* SymSpellCppPy.SymSpell, *bytes: buffer*) → None

Load internal representation from buffers, such as 'bytes' and 'memoryview'

lookup(**args*, ***kwargs*)

Overloaded function.

1. lookup(self: SymSpellCppPy.SymSpell, input: str, verbosity: SymSpellCppPy.Verbosity) -> List[SymSpellCppPy.SuggestItem]

   Find suggested spellings for a given input word, using the maximum edit distance specified during construction of the SymSpell dictionary.

2. lookup(self: SymSpellCppPy.SymSpell, input: str, verbosity: SymSpellCppPy.Verbosity, max_edit_distance: int) -> List[SymSpellCppPy.SuggestItem]

   Find suggested spellings for a given input word, using the maximum edit distance provided to the function.

3. lookup(self: SymSpellCppPy.SymSpell, input: str, verbosity: SymSpellCppPy.Verbosity, max_edit_distance: int, include_unknown: bool) -> List[SymSpellCppPy.SuggestItem]

   Find suggested spellings for a given input word, using the maximumedit distance provided to the function and include input word in suggestions if no words within edit distance found.

4. lookup(self: SymSpellCppPy.SymSpell, input: str, verbosity: SymSpellCppPy.Verbosity, max_edit_distance: int = 2, include_unknown: bool = False, transfer_casing: bool = False) -> List[SymSpellCppPy.SuggestItem]

   Find suggested spellings for a given input word, using the maximum edit distance provided to the function and include input word in suggestions if no words within edit distance found & preserve transfer casing.

lookup_compound(**args*, ***kwargs*)

Overloaded function.

1. lookup_compound(self: SymSpellCppPy.SymSpell, input: str) -> List[SymSpellCppPy.SuggestItem]

   LookupCompound supports compound-aware automatic spelling correction of multi-word input strings with three cases:

   1. Mistakenly inserted space into a correct word led to two incorrect terms.
   2. Mistakenly omitted space between two correct words led to one incorrect combined term.
   3. Multiple independent input terms with/without spelling errors.

2. lookup_compound(self: SymSpellCppPy.SymSpell, input: str, max_edit_distance: int) -> List[SymSpellCppPy.SuggestItem]

   LookupCompound supports compound-aware automatic spelling correction of multi-word input strings with three cases:

   1. Mistakenly inserted space into a correct word led to two incorrect terms.
   2. Mistakenly omitted space between two correct words led to one incorrect combined term.
   3. Multiple independent input terms with/without spelling errors.

3. lookup_compound(self: SymSpellCppPy.SymSpell, input: str, max_edit_distance: int, transfer_casing: bool) -> List[SymSpellCppPy.SuggestItem]

   LookupCompound supports compound-aware automatic spelling correction of multi-word input strings with three cases:

   1. Mistakenly inserted space into a correct word led to two incorrect terms.
   2. Mistakenly omitted space between two correct words led to one incorrect combined term.
   3. Multiple independent input terms with/without spelling errors.

max_length(*self:* SymSpellCppPy.SymSpell) → int

Retrieves the maximum length of words in the dictionary.

purge_below_threshold_words(*self:* SymSpellCppPy.SymSpell) → None

Remove all below threshold words from the dictionary.

save_pickle(*self:* SymSpellCppPy.SymSpell, *filepath: str*) → None

Save internal representation to file

save_pickle_bytes(*self:* SymSpellCppPy.SymSpell) → bytes

Save internal representation to bytes

word_count(*self:* SymSpellCppPy.SymSpell) → int

Retrieves the total number of words in the dictionary.

word_segmentation(**args*, ***kwargs*)

Overloaded function.

1. word_segmentation(self: SymSpellCppPy.SymSpell, input: str) -> SymSpellCppPy.Info

WordSegmentation divides a string into words by inserting missing spaces at the appropriate positions. Misspelled words are corrected and do not affect segmentation. Existing spaces are allowed and considered for optimum segmentation.

2. word_segmentation(self: SymSpellCppPy.SymSpell, input: str, max_edit_distance: int) -> SymSpellCppPy.Info

   WordSegmentation divides a string into words by inserting missing spaces at the appropriate positions. Misspelled words are corrected and do not affect segmentation. Existing spaces are allowed and considered for optimum segmentation.

3. word_segmentation(self: SymSpellCppPy.SymSpell, input: str, max_edit_distance: int, max_segmentation_word_length: int) -> SymSpellCppPy.Info

   WordSegmentation divides a string into words by inserting missing spaces at the appropriate positions. Misspelled words are corrected and do not affect segmentation. Existing spaces are allowed and considered for optimum segmentation.

class SymSpellCppPy.**Verbosity**

Bases: pybind11_object

Members:

TOP
[] Top suggestion with the highest term frequency of the suggestions of smallest edit distance found.

CLOSEST
[] All suggestions of smallest edit distance found, the suggestions are ordered by term frequency.

ALL
[] All suggestions <= maxEditDistance, the suggestions are ordered by edit distance, then by term frequency (highest first)

ALL = <Verbosity.ALL: 2>

CLOSEST = <Verbosity.CLOSEST: 1>

TOP = <Verbosity.TOP: 0>

property name

property value

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX