chirptext

Le Tuan Anh

Oct 04, 2022

CONTENTS:

ChirpText, an open source and free software, is a collection of text processing tools for Python.

It is not meant to be a powerful tank like the popular NTLK but a small package which you can pip-install anywhere and write a few lines of code to process textual data.

CHAPTER

ONE

MAIN FEATURES

- Parse Japanese text (Does not require `mecab-python3` package even on Windows, only a binary release (i.e. `mecab.exe`) is required)
- Built-in "lite" text annotation formats (TTL/CSV and TTL/JSON)
- Helper functions and useful data for processing English, Japanese, Chinese and Vietnamese.
- Enhanced `open()` function that support common text-based and binary-based format (txt, gz, csv, tsv, json, etc.)
- Quick text-based report generation
- Application configuration management which can make educated guess about config files' whereabouts
- **((Experimental)** Web fetcher with responsible web crawling ethics (support caching out of the box)
- **(Experimental)** Console application template

CHAPTER

TWO

INSTALLATION

Chirptext is available on PyPI and can be installed using `pip install`

```
python install chirptext
```

Note: chirptext library does not support Python 2 anymore. Please update to Python 3 to use this package.

CHAPTER

THREE

SAMPLE CODES

3.1 Using MeCab on Windows

You can download mecab binary package from http://taku910.github.io/mecab/#download and install it. After installed you can try:

```
>>> from chirptext import deko
>>> sent = deko.parse('')
>>> sent.tokens
[[(-/*/*|||)], [(-//*|||)], [(-/*/*|||)], [(-*/*/*|||)], [(-/*/*|||)], [EOS(-//|||)]]
>>> sent.words
['', '', '', '', '']
>>> sent[0].pos
''
>>> sent[0].root
''
>>> sent[0].reading
''
```

If you installed MeCab to a custom location, for example `C:\mecab\bin\mecab.exe`, try

```
>>> deko.set_mecab_bin("C:\\mecab\\bin\\mecab.exe")
>>> deko.get_mecab_bin()
'C:\\mecab\\bin\\mecab.exe'

# Just that & now you can use mecab
>>> deko.parse('').words
['', '', '', '']
```

3.2 Convenient IO APIs

```
>>> from chirptext import chio
>>> chio.write_tsv('data/test.tsv', [['a', 'b'], ['c', 'd']])
>>> chio.read_tsv('data/tes.tsv')
[['a', 'b'], ['c', 'd']]

>>> chio.write_file('data/content.tar.gz', 'Support writing to .tar.gz file')
>>> chio.read_file('data/content.tar.gz')
```

(continues on next page)

(continued from previous page)

```
'Support writing to .tar.gz file'

>>> for row in chio.read_tsv_iter('data/test.tsv'):
...     print(row)
...
['a', 'b']
['c', 'd']
```

3.2.1 Common Recipes

Web fetcher

```
from chirptext import WebHelper

web = WebHelper('~/tmp/webcache.db')
data = web.fetch('https://letuananh.github.io/test/data.json')
data
>>> b'{ "name": "Kungfu Panda" }\n'
data_json = web.fetch_json('https://letuananh.github.io/test/data.json')
data_json
>>> {'name': 'Kungfu Panda'}
```

Using Counter

```
from chirptext import Counter, TextReport
from chirptext.leutile import LOREM_IPSUM

ct = Counter()
vc = Counter()  # vowel counter
for char in LOREM_IPSUM:
    if char == ' ':
        continue
    ct.count(char)
    vc.count("Letters")
    if char in 'auieo':
        vc.count("Vowels")
    else:
        vc.count("Consonants")
vc.summarise()
ct.summarise(byfreq=True, limit=5)
```

Output

```
Letters: 377
Consonants: 212
Vowels: 165
i: 42
e: 37
t: 32
o: 29
a: 29
```

Sample TextReport

```
# a string report
rp = TextReport()  # by default, TextReport will write to standard output, i.e. terminal
rp = TextReport(TextReport.STDOUT)  # same as above
rp = TextReport('~/tmp/my-report.txt')  # output to a file
rp = TextReport.null()  # ouptut to /dev/null, i.e. nowhere
rp = TextReport.string()  # output to a string. Call rp.content() to get the string
rp = TextReport(TextReport.STRINGIO)  # same as above

# TextReport will close the output stream automatically by using the with statement
with TextReport.string() as rp:
    rp.header("Lorem Ipsum Analysis", level="h0")
    rp.header("Raw", level="h1")
    rp.print(LOREM_IPSUM)
    rp.header("Top 5 most common letters")
    ct.summarise(report=rp, limit=5)
    print(rp.content())
```

Output

```
+-------------------------------------------------------------------------------
| Lorem Ipsum Analysis
+-------------------------------------------------------------------------------

Raw
-------------------------------------------------------------
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor␣
↪incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud␣
↪exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure␣
↪dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.␣
↪Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt␣
↪mollit anim id est laborum.

Top 5 most common letters
-------------------------------------------------------------
i: 42
e: 37
```

(continues on next page)

(continued from previous page)

```
t: 32
o: 29
a: 29
```

3.2.2 API Reference

An overview of `chirptext` modules.

Chirp Text - Minimalist Text Processing Library

Enhanced IO module

Chirptext's enhanced IO functions

`chirptext.chio.`**`is_file`**(*path*)

Check if path is a path to an existing file

`chirptext.chio.`**`iter_csv_stream`**(*input_stream*, *fieldnames=None*, *sniff=False*, **args*, ***kwargs*)

Read CSV content as a table (list of lists) from an input stream

`chirptext.chio.`**`process_file`**(*path*, *processor*, *encoding='utf-8'*, *mode='rt'*, **args*, ***kwargs*)

Process a text file's content. If the file name ends with .gz, read it as gzip file

`chirptext.chio.`**`read`**(*path*, *encoding='utf-8'*, **args*, ***kwargs*)

Read text file content. If the file name ends with .gz, read it as gzip file. If mode argument is provided as 'rb', content will be read as byte stream. By default, content is read as text (string).

Read content as text >>> txt = chio.read_file("sample.txt") # Read content as binary (bytes) >>> bin = chio.read_file("sample.dat.gz", mode="rb")

Parameters

`encoding` – defaulted to UTF-8. Will be ignored if reading mode is 'rb'

`chirptext.chio.`**`read_csv`**(*path*, *fieldnames=None*, *sniff=True*, *encoding='utf-8'*, **args*, ***kwargs*)

Read CSV rows as table from a file. By default, csv.reader() will be used any output will be a list of lists. If fieldnames is provided, DictReader will be used and output will be list of OrderedDict instead. CSV sniffing (dialect detection) is enabled by default, set sniff=False to switch it off.

`chirptext.chio.`**`read_csv_iter`**(*path*, *fieldnames=None*, *sniff=True*, *mode='rt'*, *encoding='utf-8'*, **args*, ***kwargs*)

Iterate through CSV rows in a file. By default, csv.reader() will be used any output will be a list of lists. If fieldnames is provided, DictReader will be used and output will be list of OrderedDict instead. CSV sniffing (dialect detection) is enabled by default, set sniff=False to switch it off.

`chirptext.chio.`**`read_file`**(*path*, *encoding='utf-8'*, **args*, ***kwargs*)

Read text file content. If the file name ends with .gz, read it as gzip file. If mode argument is provided as 'rb', content will be read as byte stream. By default, content is read as text (string).

Read content as text >>> txt = chio.read_file("sample.txt") # Read content as binary (bytes) >>> bin = chio.read_file("sample.dat.gz", mode="rb")

Parameters

`encoding` – defaulted to UTF-8. Will be ignored if reading mode is 'rb'

chirptext.chio.**write**(*path*, *content*, *mode=None*, *encoding='utf-8'*)

Write content to a file. If the path ends with .gz, gzip will be used.

chirptext.chio.**write_csv**(*path*, *rows*, *dialect='excel'*, *fieldnames=None*, *quoting=1*, *extrasaction='ignore'*, *encoding='utf-8'*, *newline=''*, **args*, ***kwargs*)

Write rows data to a CSV file (with or without fieldnames)

By default content will be written in excel-csv dialect. This can be changed by using the optional argument dialect.

chirptext.chio.**write_file**(*path*, *content*, *mode=None*, *encoding='utf-8'*)

Write content to a file. If the path ends with .gz, gzip will be used.

chirptext.chio.**write_tsv**(*path*, *rows*, **args*, ***kwargs*)

Write rows data in tab-separated values (TSV) format

By default content will be written in excel-tab dialect. This can be changed by using the optional argument dialect.

Text annotation (TTL) module

Text Annotation (texttaglib - TTL) module

Japanese parser

Convenient Japanese text parser that produces results in TTL format

chirptext.deko.**analyse**(*content*, *splitlines=True*, *format=None*, ***kwargs*)

Japanese text > tokenize/txt/html

chirptext.deko.**get_mecab_bin**()

Get MeCab binary location

chirptext.deko.**set_mecab_bin**(*location*)

Set MeCab binary location

Chinese character radicals

Tools for processing Chinese

class chirptext.sino.**Radical**(*idseq=''*, *radical=''*, *variants=''*, *strokes=''*, *meaning=''*, *pinyin=''*, *hanviet=''*, *hiragana=''*, *romaji=''*, *hangeul=''*, *romaja=''*, *frequency=''*, *simplified=''*, *examples=''*)

Chinese Radical Source: https://en.wikipedia.org/wiki/Kangxi_radical#Table_of_radicals

Swadesh list

Language profile: UK English

class chirptext.luke.**Word**(*ID*, *word*, *score=0*, *description=''*, *rank=0*)

Swadesh word

Vietnamese support functions

Dao Phay: A collection of tools for processing Vietnamese text using Python.

chirptext.daophay.**sorted**(*list_of_strings*)

Sort a list of Vietnamese strings

Utilities

Miscellaneous tools for text processing

class chirptext.leutile.**AppConfig**(*name*, *mode='ini'*, *working_dir='.'*, *extra_potentials=None*)

Application Configuration Helper This class supports guessing configuration file location, and reads either INI (default) or JSON format.

add_potential(**patterns*)

Add a potential config file pattern

property config

Read config automatically if required

property config_path

Path to config file

load(*file_path*)

Load configuration from a specific file

locate_config()

Locate config file

read_config(*key*, *strict=False*, ***kwargs*)

Read a config by key

Default value can be passed by using the kwarg *default*

```
>>> read_config(key, default='my value')
```

Parameters

- **key** – configuration key
- **strict** – Set to True to raise KeyError if config key was not set. Defaulted to False
- **default** – Optional kwarg to set default value when key could not be found

read_file(*file_path*)

Read a configuration file and return configuration data

class chirptext.leutile.**Counter**(*priority=None*, **args*, ***kwargs*)

Powerful counter class

get_report_order()

Keys are sorted based on report order (i.e. some keys to be shown first) Related: see sorted_by_count

class chirptext.leutile.**FileHub**(**filenames*, *working_dir='.'*, *default_mode='a'*, *ext='txt'*)

A helper class for working with multiple text reports at the same time

class chirptext.leutile.**StringTool**

Common string function

class chirptext.leutile.**Table**(*header=True*, *padding=True*, *NoneValue=None*)

A text-based table which can be used with TextReport

format()

Format table to print out

class chirptext.leutile.**Timer**(*logger=None*, *report=None*)

Measure tasks' runtime

exec_time()

Calculate run time

class chirptext.leutile.**Value**(*value=None*)

Value holder

chirptext.leutile.**hamilton_allocate**(*numbers*, *total=100*, *precision=2*)

Use largest remainder (Hamilton) method to make sure rounded percentages add up to 100 >>> hamilton_allocate((33.33, 33.33, 33.33)) [33.34, 33.33, 33.33] >>> hamilton_allocate((24.99, 24.99, 24.99, 24.99)) [25.0, 25.0, 25.0, 25.0] >>> hamilton_allocate((76.69, 20.83, 2.49)) [76.69, 20.83, 2.48] >>> hamilton_allocate([13.626332, 47.989636, 9.596008, 28.788024]) [13.63, 47.99, 9.59, 28.79]

chirptext.leutile.**header**(**msg*, *level='h1'*, *separator=' '*, *print_out=<built-in function print>*)

Print header block in text mode

chirptext.leutile.**is_number**(*s*)

Check if something is a number

class chirptext.leutile.**piter**(*iterable*)

Peep-able iterator

fetch(*value_obj=None*)

Fetch the next two values

Command-line applications

Command-line interface helper

class chirptext.cli.**CLIApp**(*desc*, *add_vq=True*, *add_tasks=True*, ***kwargs*)

A simple template for command-line interface applications

add_task(*task*, *func=None*, ***kwargs*)

Add a task parser

add_version_func(*show_version*)

Enable –version and -V to show version information

add_vq(*parser*)

Add verbose & quiet options

property logger

Lazy logger

run(*func=None*)

Run the app

chirptext.cli.**config_logging**(*args*)

Override root logger's level

chirptext.cli.**setup_logging**(*config_path*, *log_dir=None*, *force_setup=False*, *default_level=30*, *silent=True*)

Try to load logging configuration from a file. Set level to INFO if failed.

Parameters

- **config_path** – Path to the logging config file (JSON)
- **log_dir** – Path to log output directory. When log_dir is not None and the directory does not exist, it will be created automatically.

Python data mapping functions

Data mapping functions

class chirptext.anhxa.**TypedJSONDecoder**(*type_map=None*, ***kwargs*)

class chirptext.anhxa.**TypedJSONEncoder**(**args*, *type_map=None*, ***kwargs*)

default(*obj*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

class chirptext.anhxa.**TypelessSONEncoder**(**args*, *type_map=None*, ***kwargs*)

default(*obj*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
```

(continues on next page)

(continued from previous page)

```
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

chirptext.anhxa.**dumps**(*obj*, **args*, ***kwargs*)

Typeless dump an object to json string

chirptext.anhxa.**flex_update_obj**(*source*, *target*, *__silent*, **fields*, ***field_map*)

Pull data from source to target. Target's __dict__ (object data) will be used by default. Otherwise, it'll be treated as a dictionary

chirptext.anhxa.**to_dict**(*obj*, **fields*, ***field_map*)

Convert an object into a dictionary

chirptext.anhxa.**to_obj**(*cls*, *obj_data=None*, **fields*, ***field_map*)

Use obj_data (dict-like) to construct an object of type cls prioritize obj_dict when there are conficts

3.3 Useful links

- Chirptext source code: https://github.com/letuananh/chirptext/
- Chirptext documentation: https://chirptext.readthedocs.io/
- Chirptext on PyPI: https://pypi.org/project/chirptext/

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

P

R

S

T

V

W