vocabulary Documentation

Release 10

Tasdik Rahman

May 14, 2017

Contents

A Command line Magician in the form of a module

Contents:

CHAPTER 1

Introduction

For a given word, using `Vocabulary`, you can get it's

- Meaning
- Synonyms
- Antonyms
- Part of speech : whether the word is a `noun`, `interjection` or an `adverb` et el
- Translate : Translate a phrase from a source language to the desired language.
- Usage example : a quick example on how to use the word in a sentence
- Pronunciation
- Hyphenation : shows the particular stress points(if any)

Features

- Written in uncomplicated `Python`
- Returns `JSON` objects
- Minimum dependencies (just uses requests)
- Easy to install
- A decent substitute to `Wordnet`(well almost!) Wanna see? Here is a *small comparison*
- Stupidly easy to use
- Fast!
- Supports
  - both, `python2.*` and `python3.*`
  - Works on Mac, Linux and Windows

How does it work

Under the hood, it makes use of 4 awesome API's to give you consistent results. The API's being

- Wordnik
- Glosbe
- BighugeLabs
- Urbandict

CHAPTER 2

Wordnet Comparison

`Wordnet` is a great resource. No doubt about it! So why should you use `Vocabulary` when we already have `Wordnet` out there?

Let's say you want to find out the synonyms for the word `car`.

- Using `Wordnet`

```
>>> from nltk.corpus import wordnet
>>> syns = wordnet.synsets('car')
>>> syns[0].lemmas[0].name
'car'
>>> [s.lemmas[0].name for s in syns]
['car', 'car', 'car', 'car', 'cable_car']

>>> [l.name for s in syns for l in s.lemmas]
['car', 'auto', 'automobile', 'machine', 'motorcar', 'car', 'railcar', 'railway_car',
↪'railroad_car', 'car', 'gondola', 'car', 'elevator_car', 'cable_car', 'car']
```

- Doing the same using `Vocabulary`

```
>>> from vocabulary import Vocabulary as vb
>>> vb.synonym("car")
'[{"seq": 0, "text": "automotive"}, {"seq": 1, "text": "motor"}, {"seq": 2, "text":
↪"wagon"}, {"seq": 3, "text": "cart"}, {"seq": 4, "text": "automobile"}]'
>>> ## load the json data
>>> car_synonyms = json.loads(vb.synonym("car"))
>>> type(car_synonyms)
<class 'list'>
>>>
```

So there you go. You get the data in an easy `JSON` format.

You can go on comparing for the other methods too.

CHAPTER 3

Installation

Option 1: installing through pip (Suggested way)

pypi package link

```
$ pip install vocabulary
```

If you are behind a proxy

```
$ pip --proxy [username:password@]domain_name:port install vocabulary
```

Note: If you get `command not found` then

```
$ sudo apt-get install python-pip
```

should fix that

Option 2: Installing from source

```
$ git clone https://github.com/prodicus/vocabulary.git
$ cd vocabulary/
$ pip install -r requirements.txt
$ python setup.py install
```

Upgrade

You can update to the latest version by doing a

```
$ pip install --upgrade vocabulary
```

Uninstalling

```
$ pip uninstall vocabulary
```

CHAPTER 4

Usage Examples

A Simple demonstration of the module

```
## Importing the module
>>> from vocabulary.vocabulary import Vocabulary as vb

## Extracting "Meaning"
>>> vb.meaning("hillbilly")
'[{"text": "Someone who is from the hills; especially from a rural area, with a␣
↪connotation of a lack of refinement or sophistication.", "seq": 0}, {"text":
↪"someone who is from the hills", "seq": 1}, {"text": "A white person from the rural␣
↪southern part of the United States.", "seq": 2}]'
>>>

## "Synonym"
>>> vb.synonym("hurricane")
'[{"text": "storm", "seq": 0}, {"text": "tropical cyclone", "seq": 1}, {"text":
↪"typhoon", "seq": 2}, {"text": "gale", "seq": 3}]'
>>>

## "Antonym"
>>> vb.antonym("respect")
'[{"text": "disesteem"}, {"text": "disrespect"}]'
>>> vb.antonym("insane")
'[{"text": "sane"}]'

## "Part of Speech"
>>> vb.part_of_speech("hello")
'[{"text": "interjection", "example": "greeting", "seq": 0}, {"text": "verb-
↪intransitive", "example": "To call.", "seq": 1}]'
>>>

## "Usage Examples"
>>> vb.usage_example("chicanery")
'[{"text": "The Bush Administration is now the commander-in-theif (lower-case␣
↪intentional) thanks to their chicanery.", "seq": 0}]'
```

```
>>>

## "Pronunciation"
>>> vb.pronunciation("hippopotamus")
'[{'raw': '(hĭp-pŏt-ms)', 'rawType': 'ahd-legacy', 'seq': 0}, {'raw': 'HH IH2 P AH0 P␣
↪AA1 T AH0 M AH0 S', 'rawType': 'arpabet', 'seq': 1}]'
>>>

## "Hyphenation"
>>> vb.hyphenation("hippopotamus")
'[{"text": "hip", "type": "secondary stress", "seq": 0}, {"text": "po", "seq": 1}, {
↪"text": "pot", "type": "stress", "seq": 2}, {"text": "a", "seq": 3}, {"text": "mus",
↪ "seq": 4}]'
>>> vb.hyphenation("amazing")
'[{"text": "a", "seq": 0}, {"text": "maz", "type": "stress", "seq": 1}, {"text": "ing
↪", "seq": 2}]'
>>>

## "Translate"
>>> vb.translate("bread", "en","fra")
'[{"seq": 0, "text": "pain"}, {"seq": 1, "text": "paner"}, {"seq": 2, "text": "pognon
↪"}, {"seq": 3, "text": "fric"}, {"seq": 4, "text": "bl\\u00e9"}]'
>>> vb.translate("goodbye", "en","es")
'[{"seq": 0, "text": "hasta luego"}, {"seq": 1, "text": "vaya con Dios"}, {"seq": 2,
↪"text": "despedida"}, {"seq": 3, "text": "adi\\u00f3s"}, {"seq": 4, "text": "vaya␣
↪con dios"}, {"seq": 5, "text": "hasta la vista"}, {"seq": 6, "text": "nos vemos"}, {
↪"seq": 7, "text": "adios"}, {"seq": 8, "text": "hasta pronto"}]'
>>>

## "Response Formatting"
>>> vb.antonym("love", format="dict")
'{"text": "hate"}''
>>> vb.antonym("love", format="list")
["hate"]
>>> vb.part_of_speech("code", format="dict")
{0: {"text": "noun", "example": "A systematically arranged and comprehensive␣
↪collection of laws."}}
>>> vb.part_of_speech("code", format="list")
[["noun", "A systematically arranged and comprehensive collection of laws."]]
```

CHAPTER 5

Help

If you need to see the usage for any of the methods, do a

```
>>> from vocabulary import Vocabulary as vb
>>> help(vb.translate)
Help on function translate in module vocabulary.vocabulary:

translate(phrase, source_lang, dest_lang)
    Gets the translations for a given word, and returns possibilites as a list
    Calls the glosbe API for getting the translation

    <source_lang> and <dest_lang> languages should be specifed in 3-letter ISO 639-3␣
↪format,
    although many 2-letter codes (en, de, fr) will work.

    See http://en.wikipedia.org/wiki/List_of_ISO_639-3_codes for full list.

    :param phrase:  word for which translation is being found
    :param source_lang: Translation from language
    :param dest_lang: Translation to language
    :returns: returns a json object
(END)
```

and so on for other functions

CHAPTER 6

Contributing

1. Fork it.
2. Clone it

create a virtualenv

```
$ virtualenv develop              # Create virtual environment
$ source develop/bin/activate     # Change default python to virtual one
(develop)$ git clone https://github.com/prodicus/vocabulary.git
(develop)$ cd vocabulary
(develop)$ pip install -r requirements.txt  # Install requirements for 'Vocabulary'␣
↪in virtual environment
```

Or, if `virtualenv` is not installed on your system:

```
$ wget https://raw.github.com/pypa/virtualenv/master/virtualenv.py
$ python virtualenv.py develop    # Create virtual environment
$ source develop/bin/activate     # Change default python to virtual one
(develop)$ git clone https://github.com/prodicus/vocabulary.git
(develop)$ cd vocabulary
(develop)$ pip install -r requirements.txt  # Install requirements for 'Vocabulary'␣
↪in virtual environment
```

3. Create your feature branch (`$ git checkout -b my-new-awesome-feature`)
4. Commit your changes (`$ git commit -am 'Added <xyz> feature'`)
5. Run tests

```
(develop) $ ./tests.py -v
```

Conform to PEP8 and if everything is running fine, integrate your feature

6. Push to the branch (`$ git push origin my-new-awesome-feature`)
7. Create new Pull Request

Hack away!

To do

- [X] Add translate module
- [X] Add an option like *JSON=False* or *JSON=True* where the former returns a list object

Tests

Running the test cases

```
$ ./tests.py -v
test_antonym_ant_key_error (tests.tests.TestModule) ... ok
test_antonym_found (tests.tests.TestModule) ... ok
test_antonym_not_found (tests.tests.TestModule) ... ok
test_hyphenation_found (tests.tests.TestModule) ... ok
test_hyphenation_not_found (tests.tests.TestModule) ... ok
test_meaning_found (tests.tests.TestModule) ... ok
test_meaning_key_error (tests.tests.TestModule) ... ok
test_meaning_not_found (tests.tests.TestModule) ... ok
test_partOfSpeech_found (tests.tests.TestModule) ... ok
test_partOfSpeech_not_found (tests.tests.TestModule) ... ok
test_pronunciation_found (tests.tests.TestModule) ... ok
test_pronunciation_not_found (tests.tests.TestModule) ... ok
test_respond_as_dict_1 (tests.tests.TestModule) ... ok
test_respond_as_dict_2 (tests.tests.TestModule) ... ok
test_respond_as_dict_3 (tests.tests.TestModule) ... ok
test_respond_as_list_1 (tests.tests.TestModule) ... ok
test_respond_as_list_2 (tests.tests.TestModule) ... ok
test_respond_as_list_3 (tests.tests.TestModule) ... ok
test_synonynm_empty_list (tests.tests.TestModule) ... ok
test_synonynm_found (tests.tests.TestModule) ... ok
test_synonynm_not_found (tests.tests.TestModule) ... ok
test_synonynm_tuc_key_error (tests.tests.TestModule) ... ok
test_translate_empty_list (tests.tests.TestModule) ... ok
test_translate_found (tests.tests.TestModule) ... ok
test_translate_not_found (tests.tests.TestModule) ... ok
test_translate_tuc_key_error (tests.tests.TestModule) ... ok
test_usageExample_empty_list (tests.tests.TestModule) ... ok
test_usageExample_found (tests.tests.TestModule) ... ok
test_usageExample_not_found (tests.tests.TestModule) ... ok

----------------------------------------------------------------------
Ran 29 tests in 0.015s

OK
```

Discuss

Join us on our Gitter channel if you want to chat or if you have any questions.

Building the docs

Install the *Sphinx* by doing a *$ pip install requirements-dev.txt*

```
$ make html
```

Contributors

- Huge shoutout to @tenorz007 for adding the ability to return the API response as different data structures.
- Thanks to Anton Relin for adding the *translate()* module
- A big shout out to all the contributers

CHAPTER 7

Changelog

0.0.4

- `JSON` inconsistency fixed for the methods
  - `Vocabulary.hyphenation()`
  - `Vocabulary.part_of_speech()`
  - `Vocabulary.meaning()`

0.0.5

New in version 0.0.5.

- Added `Vocabulary.translate()`
- Improved Documentation
- Minor bug fixes

1.0.0

New in version 1.0.0.

- Added support for specifying response format
- Updated `Vocabulary.pronunciation`, `Vocabulary.antonym`, `Vocabulary.part_of_speech` to return a list of objects with appriprate index

CHAPTER 8

Known Issues

When using the method `pronunciation`

```
>>> vb.pronunciation("hippopotamus")
[{'raw': '(hĭp-pŏt-ms)', 'rawType': 'ahd-legacy', 'seq': 0}, {'raw': 'HH IH2 P AH0 P␣
↪AA1 T AH0 M AH0 S', 'rawType': 'arpabet', 'seq': 1}]
>>> type(vb.pronunciation("hippopotamus"))
<class 'list'>
>>> json.dumps(vb.pronunciation("hippopotamus"))
'[{"raw": "(h\\u012dp\\u02cc\\u0259-p\\u014ft\\u02c8\\u0259-m\\u0259s)", "rawType":
↪"ahd-legacy", "seq": 0}, {"raw": "HH IH2 P AH0 P AA1 T AH0 M AH0 S", "rawType":
↪"arpabet", "seq": 1}]'
>>>
```

You are being returned a `list` object instead of a `JSON` object. When returning the latter, there are some `unicode` issues. A fix for this will be released soon.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search