geograpy3

Sep 21, 2020

Contents:

CHAPTER 1

geograpy package

1.1 Submodules

1.2 geograpy.extraction module

class geograpy.extraction.**Extractor**(*text=None*, *url=None*, *debug=False*)

Bases: object

Extract geo context for text or from url

find_entities(*labels=['GPE', 'GSP', 'PERSON', 'ORGANIZATION']*)

Find entities with the given labels set self.places and returns it :param labels: Labels: The labels to filter

Returns List of places

Return type list

find_geoEntities()

Find geographic entities

Returns List of places

Return type list

set_text()

Setter for text

split()

simpler regular expression splitter with not entity check

hat tip: https://stackoverflow.com/a/1059601/1497139

1.3 geograpy.labels module

Created on 2020-09-10

@author: wf

class geograpy.labels.**Labels**
Bases: object

NLTK labels

default = ['GPE', 'GSP', 'PERSON', 'ORGANIZATION']

geo = ['GPE', 'GSP']

1.4 geograpy.locator module

The locator module allows to get detailed city information including the region and country of a city from a location string.

Examples for location strings are:

Amsterdam, Netherlands Vienna, Austria Vienna, IL Paris - Texas Paris TX

the locator will lookup the cities and try to disambiguate the result based on the country or region information found.

The results in string representationa are:

Amsterdam (NH(North Holland) - NL(Netherlands)) Vienna (9(Vienna) - AT(Austria)) Vienna (IL(Illinois) - US(United States)) Paris (TX(Texas) - US(United States)) Paris (TX(Texas) - US(United States))

Each city returned has a city.region and city.country attribute with the details of the city.

Created on 2020-09-18

@author: wf

class geograpy.locator.**City**
Bases: object

a single city as an object

static fromGeoLite2(*record*)

class geograpy.locator.**Country**
Bases: object

a country

static fromGeoLite2(*record*)
create a country from a geolite2 record

static fromPyCountry(*pcountry*)

Parameters **pcountry** (*PyCountry*) – a country as gotten from pycountry

Returns the country

Return type *Country*

class geograpy.locator.**Locator**(*db_file=None*, *correctMisspelling=False*, *debug=False*)
Bases: object

location handling

cities_for_name(*city_name*)
find cities with the given city_name

Parameters **city_name** (*string*) – the potential name of a city

Returns a list of city records

correct_country_misspelling(*name*)
correct potential misspellings :param name: the name of the country potentially misspelled :type name: string

Returns correct name of unchanged

Return type string

db_has_data()
check whether the database has data / is populated

Returns True if the cities table exists and has more than one record

Return type boolean

disambiguate(*country*, *regions*, *cities*)
try determining country, regions and city from the potential choices

Parameters

- **country** (Country) – a matching country found
- **regions** (*list*) – a list of matching Regions found
- **cities** (*list*) – a list of matching cities found

Returns the found city or None

Return type *City*

getCountry(*name*)
get the country for the given name :param name: the name of the country to lookup :type name: string

Returns the country if one was found or None if not

Return type country

getGeolite2Cities()
get the Geolite2 City-Locations as a list of Dicts

Returns a list of Geolite2 City-Locator dicts

Return type list

static getInstance(*correctMisspelling=False*, *debug=False*)
get the singleton instance of the Locator. If parameters are changed on further calls the initial parameters will still be in effect since the original instance will be returned!

Parameters

- **correctMispelling** (*bool*) – if True correct typical misspellings
- **debug** (*bool*) – if True show debug information

isAmbiguousPrefix(*name*)
check if the given name is an ambiguous prefix

Parameters **name** (*string*) – the city name to check

Returns True if this is a known prefix that is ambigous that is there is also a city with such a name

Return type bool

isISO(*s*)
check if the given string is an ISO code

Returns True if the string is an ISO Code

Return type bool

isPrefix(*name*, *level*)
check if the given name is a city prefix at the given level

Parameters

- **name** (*string*) – the city name to check
- **level** (*int*) – the level on which to check (number of words)

Returns True if this is a known prefix of multiple cities e.g. "San", "New", "Los"

Return type bool

is_a_country(*name*)
check if the given string name is a country

Parameters **name** (*string*) – the string to check

Returns if pycountry thinks the string is a country

Return type True

locate(*places*)
locate a city, region country combination based on the places information

Parameters **places** (*list*) – a list of place tokens e.g. "Vienna, Austria"

Returns a city with country and region details

Return type *City*

locator = None

places_by_name(*place_name*, *column_name*)
get places by name and column :param place_name: the name of the place :type place_name: string :param column_name: the column to look at :type column_name: string

populate_Cities(*sqlDB*)
populate the given sqlDB with the Geolite2 Cities

Parameters **sqlDB** (*SQLDB*) – the SQL database to use

populate_PrefixAmbiguities(*sqlDB*)
create a table with ambiguous prefixes

Parameters **sqlDB** (*SQLDB*) – the SQL database to use

populate_PrefixTree(*sqlDB*)
calculate the PrefixTree info

Parameters **sqlDb** – the SQL Database to use

Returns the prefix tree

Return type *PrefixTree*

populate_db(*force=False*)
populate the cities SQL database which caches the information from the GeoLite2-City-Locations.csv file

regions_for_name(*region_name*)
get the regions for the given region_name (which might be an ISO code)

Parameters **region_name** (*string*) – region name

Returns the list of cities for this region

Return type list

class geograpy.locator.**Region**
Bases: object

a Region (Subdivision)

static fromGeoLite2(*record*)
create a region from a Geolite2 record

Parameters **record** (*dict*) – the records as returned from a Query

Returns the corresponding region information

Return type *Region*

1.5 geograpy.places module

class geograpy.places.**PlaceContext**(*place_names*, *setAll=True*)
Bases: *geograpy.locator.Locator*

Adds context information to a place name

get_region_names(*country_name*)

setAll()
Set all context information

set_cities()
set the cities information

set_countries()
get the country information from my places

set_other()

set_regions()

1.6 geograpy.prefixtree module

Created on 2020-09-20

@author: wf

class geograpy.prefixtree.**PrefixTree**
Bases: object

prefix analysis and search

see http://p-nand-q.com/python/data-types/general/tries.html

add(*name*)
add the given name to the prefix Tree

Parameters **name** (*string*) – the name to add

add2Table(*prefix*, *prefixStr*, *table*, *level*)
recursively add prefix tree entries to a table

Parameters

- **prefix** (*dict*) – the dictionary to start with
- **prefixStr** (*string*) – the prefix string up to this level
- **table** (*list*) – a "flat" list of dicts as a table
- **level** (*int*) – the level (length of word sequence) on which to add

countStartsWith(*namePrefix*)
count how many entries start with the given namePrefix

Parameters **namePrefix** (*string*) – the prefix to check

getCount()
get my total count

Returns the total number of entries

Return type int

getWords(*name*)
split the given name into words

Parameters **name** (*string*) – the name to split

Returns a list of words

Return type list

store(*sqlDB*)
store my prefix information to the given SQL database

Parameters **sqlDB** (*SQLDB*) – the SQL database to use for storing

1.7 geograpy.utils module

geograpy.utils.**fuzzy_match**(*s1*, *s2*, *max_dist=0.8*)
Fuzzy match the given two strings with the given maximum distance :param s1: string: First string :param s2: string: Second string :param max_dist: float: The distance - default: 0.8

Returns jellyfish jaro_winkler_similarity based on https://en.wikipedia.org/wiki/Jaro-Winkler_distance

Return type float

geograpy.utils.**remove_non_ascii**(*s*)
Remove non ascii chars from the given string :param s: string: The string to remove chars from

Returns The result string with non-ascii chars removed

Return type string

Hat tip: http://stackoverflow.com/a/1342373/2367526

1.8 Module contents

geograpy.**get_geoPlace_context**(*url=None*, *text=None*, *debug=False*)
Get a place context for a given text with information about country, region, city and other based on NLTK Named Entities having the Geographic(GPE) label.

Parameters

- **url** (*String*) – the url to read text from (if any)
- **text** (*String*) – the text to analyze
- **debug** (*boolean*) – if True show debug information

Returns PlaceContext: the place context

Return type places

geograpy.**get_place_context**(*url=None, text=None, labels=['GPE', 'GSP', 'PERSON', 'ORGANIZATION'], debug=False*)

Get a place context for a given text with information about country, region, city and other based on NLTK Named Entities in the label set Geographic(GPE), Person(PERSON) and Organization(ORGANIZATION).

Parameters

- **url** (*String*) – the url to read text from (if any)
- **text** (*String*) – the text to analyze
- **debug** (*boolean*) – if True show debug information

Returns PlaceContext: the place context

Return type pc

geograpy.**locate**(*location*, *correctMisspelling=False*, *debug=False*)

locate the given location string :param location: the description of the location :type location: string

Returns the location

Return type *Locator*

CHAPTER 2

setup module

CHAPTER 3

tests package

3.1 Submodules

3.2 tests.test_extractor module

class tests.test_extractor.**TestExtractor**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test Extractor

check(*places*, *expectedList*)
check the places for begin non empty and having at least the expected List of elements

Parameters

- **places** (*Places*) – the places to check
- **expectedList** (*list*) – the list of elements to check

setUp()
Hook method for setting up the test fixture before exercising it.

tearDown()
Hook method for deconstructing the test fixture after testing it.

testExtractorFromText()
test different texts for getting geo context information

testExtractorFromUrl()
test the extractor

testGeograpyIssue32()
test https://github.com/ushahidi/geograpy/issues/32

testGetGeoPlace()
test geo place handling

testIssue10()
test https://github.com/somnathrakshit/geograpy3/issues/10 Add ISO country code

testIssue7()
test https://github.com/somnathrakshit/geograpy3/issues/7 disambiguating countries

testIssue9()
test https://github.com/somnathrakshit/geograpy3/issues/9 [BUG]AttributeError: 'NoneType' object has no attribute 'name' on "Pristina, Kosovo"

testStackOverflow54721435()
see https://stackoverflow.com/questions/54721435/unable-to-extract-city-names-from-a-text-using-geograpypython

testStackoverflow43322567()
see https://stackoverflow.com/questions/43322567

testStackoverflow54077973()
see https://stackoverflow.com/questions/54077973/geograpy3-library-for-extracting-the-locations-in-the-text-gives-unicode

testStackoverflow54712198()
see https://stackoverflow.com/questions/54712198/not-only-extracting-places-from-a-text-but-also-other-names-in-geograp

testStackoverflow55548116()
see https://stackoverflow.com/questions/55548116/geograpy3-library-is-not-working-properly-and-give-traceback-error

testStackoverflow62152428()
see https://stackoverflow.com/questions/62152428/extracting-country-information-from-description-using-geograpy?noredirect=1#comment112899776_62152428

3.3 tests.test_locator module

Created on 2020-09-19

@author: wf

class tests.test_locator.**TestLocator**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test the Locator class from the location module

setUp()
Hook method for setting up the test fixture before exercising it.

tearDown()
Hook method for deconstructing the test fixture after testing it.

testExamples()
test examples

testGeolite2Cities()
test the locs.db cache for cities

testHasData()
check has data and populate functionality

testIsoRegexp()
test regular expression for iso codes

testWordCount()
test the word count

3.4 tests.test_places module

class tests.test_places.**TestPlaces**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test Places

setUp()
Hook method for setting up the test fixture before exercising it.

tearDown()
Hook method for deconstructing the test fixture after testing it.

testPlaces()
test places

3.5 tests.test_prefixtree module

Created on 2020-09-20

@author: wf

class tests.test_prefixtree.**TestPrefixTree**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test prefix tree algorithm

setUp()
Hook method for setting up the test fixture before exercising it.

tearDown()
Hook method for deconstructing the test fixture after testing it.

testPrefixTree()
test the prefix Tree

3.6 Module contents

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

t

Index

R

S

T