geograpy3

Jun 23, 2021

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(*delimiter=', '*)

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**(***kwargs*)
Bases: *geograpy.locator.Location*

a single city as an object

country

static fromGeoLite2(*record*)

classmethod getSamples()

region

setValue(*name*, *record*)
set a field value with the given name to the given record dicts corresponding entry or none

Parameters

- **name** (*string*) – the name of the field
- **record** (*dict*) – the dict to get the value from

class geograpy.locator.**CityList**
Bases: *geograpy.locator.LocationList*

a list of cities

classmethod fromJSONBackup()
get city list from json backup (json backup is based on wikidata query results)

Returns CityList based on the json backup

classmethod fromWikidata(*fromBackup: bool = True*, *countryIDs: list = None*, *regionIDs: list = None*)

get city list form wikidata

Parameters

- **fromBackup** (*bool*) – If True instead of querying wikidata a backup of the wikidata results is used to create the city list. Otherwise wikidata is queried for the city data. Default is True
- **countryIDs** (*list*) – List of countryWikiDataIDs. Limits the returned cities to the given countries
- **regionIDs** (*list*) – List of regionWikiDataIDs. Limits the returned cities to the given regions

Returns CityList based wikidata query results

class geograpy.locator.**Country**(*lookupSource='sqlDB'*, ***kwargs*)

Bases: *geograpy.locator.Location*

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*

classmethod getSamples()

class geograpy.locator.**CountryList**

Bases: *geograpy.locator.LocationList*

a list of countries

classmethod fromErdem()

get country list provided by Erdem Ozkol https://github.com/erdem

classmethod fromJSONBackup()

get country list from json backup (json backup is based on wikidata query results)

Returns CountryList based on the json backup

classmethod fromWikidata()

get country list form wikidata

classmethod from_sqlDb(*sqlDB*)

class geograpy.locator.**Earth**

Bases: object

radius = 6371.0

class geograpy.locator.**Location**(***kwargs*)

Bases: lodstorage.jsonable.JSONAble

Represents a Location

balltreeQueryResultToLocationList(*distances*, *indices*, *lookupListOfLocations*)

convert the given ballTree Query Result to a LocationList

Parameters

- **distances** (*list*) – array of distances
- **indices** (*list*) – array of indices
- **lookupListOfLocations** (*list*) – a list of valid locations to use for lookup

Returns a list of result Location/distance tuples

Return type list

distance(*other*) → float
calculate the distance to another Location

Parameters **other** (Location) – the other location

Returns the haversine distance in km

getLocationsWithinRadius(*lookupLocationList*, *radiusKm: float*)
Gives the n closest locations to me from the given lookupListOfLocations

Parameters

- **lookupLocationList** (LocationList) – a LocationList object to use for lookup
- **radiusKm** (*float*) – the radius in which to check (in km)

Returns a list of result Location/distance tuples

Return type list

getNClosestLocations(*lookupLocationList*, *n: int*)
Gives a list of up to n locations which have the shortest distance to me as calculated from the given listOfLocations

Parameters

- **lookupLocationList** (LocationList) – a LocationList object to use for lookup
- **n** (*int*) – the maximum number of closest locations to return

Returns a list of result Location/distance tuples

Return type list

classmethod getSamples()

static haversine(*lon1*, *lat1*, *lon2*, *lat2*)
Calculate the great circle distance between two points on the earth (specified in decimal degrees)

isKnownAs(*name*) → bool
Checks if this location is known under the given name

Parameters **name** (*str*) – name the location should be checked against

Returns True if the given name is either the name of the location or present in the labels of the location

class geograpy.locator.**LocationContext**(*countryList: geograpy.locator.CountryList*, *regionList: geograpy.locator.RegionList*, *cityList: geograpy.locator.CityList*)

Bases: object

Holds LocationLists of all hierarchy levels and provides methods to traverse through the levels

cities

cityList

countries

countryList

classmethod fromJSONBackup()
Inits a LocationContext form the JSON backup

getCities(*name: str*)
Returns all cities that are known under the given name

getCountries(*name: str*)
Returns all countries that are known under the given name

getRegions(*name: str*)
Returns all regions that are known under the given name

regionList

regions

class geograpy.locator.**LocationList**(*listName: str = None*, *clazz=None*, *tableName: str = None*)
Bases: lodstorage.jsonable.JSONAbleList

a list of locations

getBallTuple(*cache: bool = True*)
get the BallTuple=BallTree,validList of this location list

Parameters

- **cache** (*bool*) – if True calculate and use a cached version otherwise recalculate on
- **call of this function** (*every*) –

Returns a sklearn.neighbors.BallTree for the given list of locations, list: the valid list of locations list: valid list of locations

Return type BallTree,list

getLocationByID(*wikidataID: str*)
Returns the location object that corresponds to the given location

Parameters **wikidataID** – wikidataid of the location that should be returned

Returns Location object

getLocationList()
get my location list

static getURLContent(*url: str*)

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

location handling

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

Parameters **cityName** (*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

createViews(*sqlDB*)

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

db_recordCount(*tableList*, *tableName*)
count the number of records for the given tableName

Parameters

- **tableList** (*list*) – the list of table to check
- **tableName** (*str*) – the name of the table to check

Returns int: the number of records found for the table

disambiguate(*country*, *regions*, *cities*, *byPopulation=True*)
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*

getAliases()
get the aliases hashTable

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

getView()
get the view to be used

Returns the SQL view to be used for CityLookups e.g. GeoLite2CityLookup

Return type str

getWikidataCityPopulation(*sqlDB*, *endpoint=None*)

Parameters

- **sqlDB** (*SQLDB*) – target SQL database
- **endpoint** (*str*) – url of the wikidata endpoint or None if default should be used

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

Returns True if the string is an ISO Code

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

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

Parameters

- **places** (*list*) – a list of places derived by splitting a locality e.g. “San Francisco, CA”
- **to "San Francisco", "CA"** (*leads*) –

Returns a city with country and region details

Return type *City*

locator = None

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

populateFromWikidata(*sqlDB*)
populate countries and regions from Wikidata

Parameters **sqlDB** (*SQLDB*) – target SQL database

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

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

populate_Cities_FromWikidata(*sqlDB*)
populate the given sqlDB with the Wikidata Cities

Parameters **sqlDB** (*SQLDB*) – target SQL database

populate_Countries(*sqlDB*)
populate database with countries from wikiData

Parameters **sqlDB** (*SQLDB*) – target SQL database

populate_Regions(*sqlDB*)
populate database with regions from wikiData

Parameters **sqlDB** (*SQLDB*) – target SQL database

populate_Version(*sqlDB*)
populate the version table

Parameters **sqlDB** (*SQLDB*) – target SQL database

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

Parameters **force** (*bool*) – if True force a recreation of the database

readCSV(*fileName*)

recreateDatabase()
recreate my lookup database

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

static resetInstance()

class geograpy.locator.**Region**(***kwargs*)
Bases: *geograpy.locator.Location*

a Region (Subdivision)

country

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*

static fromWikidata(*record*)
create a region from a Wikidata record

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

Returns the corresponding region information

Return type *Region*

classmethod getSamples()

class geograpy.locator.**RegionList**
Bases: *geograpy.locator.LocationList*

a list of regions

classmethod fromJSONBackup()
get region list from json backup (json backup is based on wikidata query results)

Returns RegionList based on the json backup

classmethod fromWikidata()
get region list form wikidata

classmethod from_sqlDb(*sqlDB*)

geography.locator.**main**(*argv=None*)
main program.

1.5 geography.places module

class geography.places.**PlaceContext**(*place_names*, *setAll=True*)
Bases: *geography.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 geography.prefixtree module

1.7 geography.utils module

geography.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

geography.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 geograpy.wikidata module

Created on 2020-09-23

@author: wf

class geograpy.wikidata.**Wikidata**(*endpoint='https://query.wikidata.org/sparql'*)
Bases: object

Wikidata access

getCities(*region=None*, *country=None*)
get the cities from Wikidata

Parameters

- **region** – List of countryWikiDataIDs. Limits the returned cities to the given countries
- **country** – List of regionWikiDataIDs. Limits the returned cities to the given regions

getCitiesOfRegion(*regionWikidataId: str*, *limit: int*)
Queries the cities of the given region. If the region is a city state the region is returned as city. The cities are ordered by population and can be limited by the given limit attribute.

Parameters

- **regionWikidataId** – wikidata id of the region the cities should be queried for
- **limit** – Limits the amount of returned cities

Returns Returns list of cities of the given region ordered by population

getCityPopulations(*profile=True*)
get the city populations from Wikidata

Parameters **profile** (*bool*) – if True show profiling information

static getCoordinateComponents(*coordinate: str) -> (<class 'float'>*, *<class 'float'>*)
Converts the wikidata coordinate representation into its subcomponents longitude and latitude Example: 'Point(-118.25 35.05694444)' results in ('-118.25' '35.05694444')

Parameters **coordinate** – coordinate value in the format as returned by wikidata queries

Returns Returns the longitude and latitude of the given coordinate as separate values

getCountries()
get a list of countries

try query

getRegions()
get Regions from Wikidata

try query

static getValuesClause(*varName: str*, *values*, *wikidataEntities: bool = True*)
generates the SPARQL value clause for the given variable name containing the given values :param varName: variable name for the ValuesClause :param values: values for the clause :param wikidataEntities: if true the wikidata prefix is added to the values otherwise it is expected taht the given values are proper IRIs :type wikidataEntities: bool

Returns str

static getWikidataId(*wikidataURL: str*)
Extracts the wikidata id from the given wikidata URL

Parameters **wikidataURL** – wikidata URL the id should be extracted from

Returns The wikidata id if present in the given wikidata URL otherwise None

1.9 Module contents

main geograpy 3 module

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.**locateCity**(*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

checkExamples(*examples*, *countries*, *debug=False*, *check=True*)
check that the given example give results in the given countries :param examples: a list of example location strings :type examples: list :param countries: a list of expected country iso codes :type countries: list

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

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

testDelimiters()
test the delimiter statistics for names

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

testIssue15()
https://github.com/somnathrakshit/geograpy3/issues/15 test Issue 15 Disambiguate via population, gdp data

testIssue17()
test issue 17:

https://github.com/somnathrakshit/geograpy3/issues/17

[BUG] San Francisco, USA and Auckland, New Zealand should be locatable #17

testIssue19()
test issue 19

testIssue22()
https://github.com/somnathrakshit/geograpy3/issues/22

testIssue41_CountriesFromErdem()
test getting Country list from Erdem

testIssue_42_distance()
test haversine and location

testPopulation()
test adding population data from wikidata to GeoLite2 information

testProceedingsExample()
test a proceedings title Example

testStackOverflow64379688()
compare old and new geograpy interface

testStackOverflow64418919()
https://stackoverflow.com/questions/64418919/problem-retrieving-region-in-us-with-geograpy3

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

3.6 tests.test_wikidata module

Created on 2020-09-23

@author: wf

class tests.test_wikidata.**TestWikidata**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test the wikidata access for cities

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

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

testGetCitiesOfRegion()
Test getting cities based on region wikidata id

testGetCoordinateComponents()
test the splitting of coordinate components in WikiData query results

testGetWikidataId()

testLocatorWithWikiData()
test Locator

testWikidataCities()

test getting city information from wikidata

1372 Singapore 749 Beijing, China 704 Paris, France 649 Barcelona, Spain 625 Rome, Italy 616 Hong Kong 575 Bangkok, Thailand 502 Vienna, Austria 497 Athens, Greece 483 Shanghai, China

testWikidataCountries()
test getting country information from wikidata

3.7 Module contents

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

t

Index

G

H

I

L

M

P

R

S

T

W