addrmatcher

Release 0.0.2.5

Irfan, Hnin, Yun, Jedo

Dec 03, 2021

CONTENTS:

CHAPTER

ONE

INTRODUCTION

Addrmatcher is an open-source Python software for matching input string addresses to the most similar street addresses and the geo coordinates inputs to the nearest street addresses. The result provides not only the matched addresses, but also the respective country's different levels of regions for instance - in Australia, government administrative regions, statistical areas and suburbs in which the address belongs to.

The Addrmatcher library is built to work with rapidfuzz, scikit-learn, pandas, numpy and provides user-friendly output. It supports python version 3.6 and above. It runs on all popular operating systems, and quick to install and is free of charge.

In this initial release, the scope of input data and matching capability are limited to Australian addresses only. The Addrmatcher library will see the opportunity to scale the matching beyond Australia in future.

The package offers two matching capabilities

- **address-based matching** accepts string address as argument.
- **coordinate-based matching** takes geo coordinate (latitude and longitude) as input.

The development team achieved the optimal speed of matching less than one second for each address and each pair of coordinate input.

The reference dataset is built upon GNAF(Geocoded National Address File) and ASGS(Australian Statistical Geography Standard) for the Australian addresses. The package users will require to download the optimised format of reference dataset into the working direcory once the package has been installed.

1.1 Installation

```
pip install addrmatcher
```

1.2 Data Download

Once the package has been installed, the reference dataset needs to be downloaded into **the local current project working directory** prior to implementation of the package's matching functions.

In the command line interface, .. code-block:

```
addrmatcher-data aus
```

The above console script will download the dataset which is currently hosted in Github into the user's directory. addrmatcher-data takes an argument, country. By default, the country is Australia which is indicated by aus. Then

Australia address files will be downloaded. After executing the command, the 37 parquet files will be stored in directories for example /data/Australia/*.parquet.

1.3 Package and Classes Import

```
# Import the installed package
from addrmatcher import AUS, GeoMatcher

# Initialise the geo region as AUS
matcher = GeoMatcher(AUS)
```

1.4 Example - Address Matching

```
matched_address = matcher.get_region_by_address("9121, George Street, North Strathfield,␣
↪NSW 2137")
print(matched_address)

> {'SA4_NAME_2016': ['Sydney - Inner West'],
  'LGA_NAME_2016': ['Canada Bay (A)'],
  'SA3_NAME_2016': ['Canada Bay'],
  'RATIO': [100.0],
  'STATE': ['NSW'],
  'FULL_ADDRESS': ['9121 GEORGE STREET NORTH STRATHFIELD NSW 2137'],
  'SA2_NAME_2016': ['Concord West - North Strathfield'],
  'SSC_NAME_2016': ['North Strathfield'],
  'MB_CODE_2016': ['11205258900'],
  'SA1_7DIGITCODE_2016': ['1138404']}
```

1.5 Example - Coordinate Matching

```
nearest_address = matcher.get_region_by_coordinates(-29.1789874, 152.628291)
print(nearest_address)

>{'IDX': [129736],
'FULL_ADDRESS': ['3 7679 CLARENCE WAY MALABUGILMAH NSW 2460'],
'LATITUDE': [-29.17898685],
'LONGITUDE': [152.62829132],
'LGA_NAME_2016': ['Clarence Valley (A)'],
'SSC_NAME_2016': ['Baryulgil'],
'SA4_NAME_2016': ['Coffs Harbour - Grafton'],
'SA3_NAME_2016': ['Clarence Valley'],
'SA2_NAME_2016': ['Grafton Region'],
'SA1_7DIGITCODE_2016': ['1108103'],
'MB_CODE_2016': ['11205732700'],
'STREET_NAME': ['CLARENCE'],
'STREET_TYPE_CODE': ['WAY'],
'LOCALITY_NAME': ['MALABUGILMAH'],
```

(continues on next page)

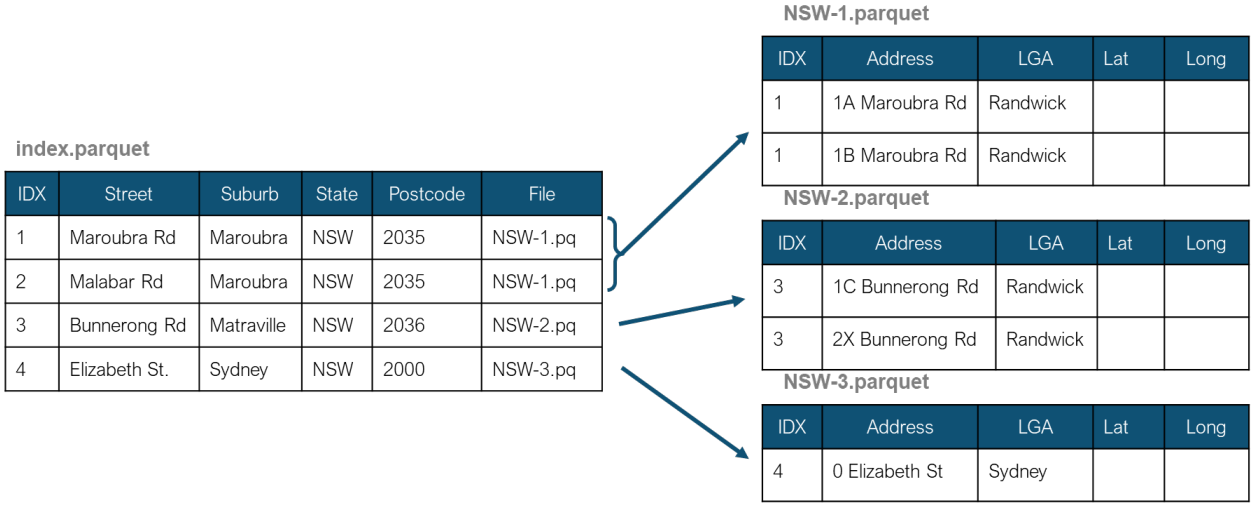
(continued from previous page)

```
'STATE': ['NSW'],
'POSTCODE': ['2460'],
'ADDRESS_DETAIL_PID': ['GANSW706638188'],
'FILE_NAME': ['NSW-10.parquet'],
'DISTANCE': [6.859565028181215e-05]}
```

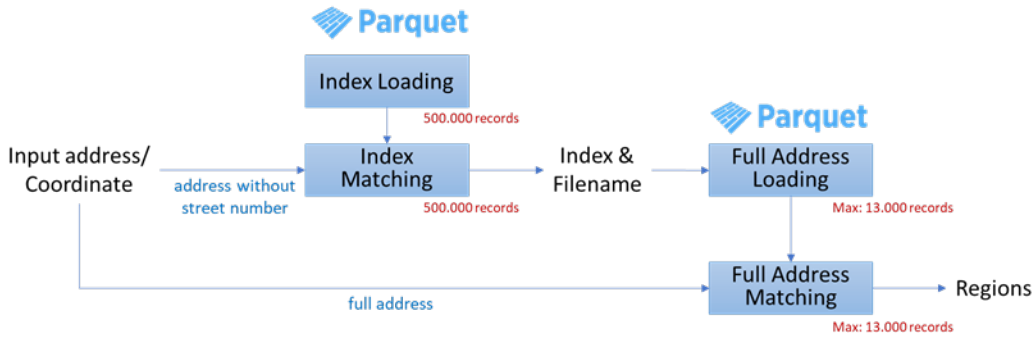
1.6 How the Address Matching Works?

1.6.1 1. Address-based matching

The idea behind the address-based matching function is comparing the similarity between two addresses. The more similar the strings are, the more likely both addresses are identical. Therefore, the package adopted the edit-distance method (Levenshtein, Jaro, and Jaro-Winkler) to quantify text similarity based on the minimum number of operations required to transform one string to the other. The package performs address matching by comparing the similarity of the input address with the reference dataset. The function then returns the address and its corresponding regional level that has the highest similarity ratio.



An index file was created to store the unique combination of the street name, locality, state, and postcode. The index file keeps the distinct physical addresses without street numbers, lot numbers, and other similar attributes. Also, the complete addresses were divided into multiple files to limit the number of rows in each file below 500,000 addresses. The index file then stores the filename of the full physical address location.

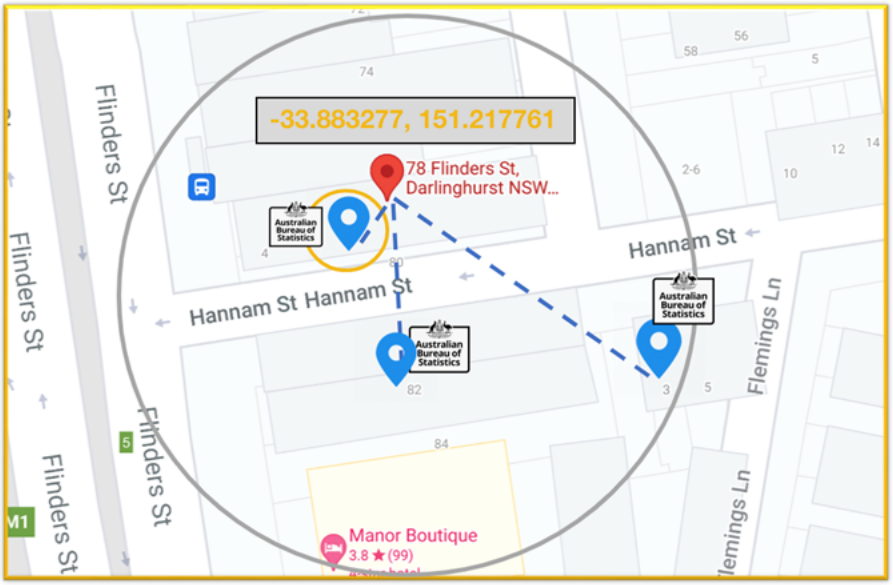


With this file structure, the package does not need to load all 15 million records and compare the input address with

the entire list of addresses. Instead, the package only needs to load the index file and match the combination of street, suburb, state and postcode from the input address with composite of those in the index file. Then, the matched combination of street, suburb, state and postcode gets the name of the respective address file to load into the memory.After that, string matching is performed between the input address and the addresses in the file. Therefore, the package only needs to load and match the small factional of the entire dataset.

1.6.2 2. Coordinate-based matching

Coordinate-based matching is distance-based matching. The matching is performed by searching for closer addresses in the GNAF dataset to the input geo-coordinates based on geo-distances.



CHAPTER

TWO

ADDRMATCHER

2.1 Hierarchies

class addrmatcher.hierarchy.**GeoHierarchy**(*country*, *name*, *coordinate_boundary=None*)

Bases: object

The GeoHierarchy class represents the structure of a country's region/area; for instance, a state or a province is the sub-region of a country.

Attributes

coordinate_boundary Return the list of coordinates as the boundary of a country.

name Return the name of the country

types Return the dictionary that contains all the defined regional structures.

Methods

add_region(region, parent_region)	Add a region as a child/sub region of another region
add_type(region, type_id[, type_name])	Add a new geographical hierarchy type, for instace: statistical area, administrative level
get_regions_by_name([region_names, ...])	Get all the relevant regions from the hierarchy based on the given parameters
get_smallest_region_boundaries()	Get the smallest regional unit
set_coordinate_boundary(min_latitude, ...)	Set or modify the coordinate boundaries of a country.

add_region(*region*, *parent_region*)

Add a region as a child/sub region of another region

Parameters

region:Region The sub region to be added as a child of the parent region

parent_region:Region The direct upper-level of the region

Examples

```
>>> country = Region("Country")
>>> state = Region("State",col_name="STATE")
>>> australia = GeoHierarchy(country,"Australia")
>>> australia.add_region(region=state, parent_region=country)
```

add_type(*region*, *type_id*, *type_name=''*)

Add a new geographical hierarchy type, for instace: statistical area, administrative level

Parameters

region:Region The smallest root region for the hierarchy. The common upper-level region name, shared with other types of hierarchies, can't be assigned as a root. For instance, administrative level and statistical area use Country or State/Province as their upper regional level

type_id:string The unique identifier for the structural type

type_name:string The name of the regional structure type

Examples

```
>>> country = Region("Country")
>>> state = Region("State",col_name="STATE")
>>> sa4 = Region("Statistica Area 4",short_name="SA4",col_name="SA4")
>>> australia = GeoHierarchy(country,"Australia")
>>> australia.add_region(region=state, parent_region=country)
>>> australia.add_region(region=sa4, parent_region=state)
>>> australia.add_type(sa4,"ASGS","Australian Statistical Geography Standard")
>>> australia.type
{'ASGS': 'Australian Statistical Geography Standard'}
```

property coordinate_boundary

Return the list of coordinates as the boundary of a country. The format is [minimum latitude, maximum latitude, minimum longitude, maximum longitude]

Returns

string The coordinate boundary of a country

Examples

```
>>> australia = GeoHierarchy(country,"Australia")
>>> australia.set_coordinate_boundary(-43.58301104, -9.23000371,
                                      96.82159219, 167.99384663)
>>> australia.coordinate_boundary
[-43.58301104, -9.23000371, 96.82159219, 167.99384663]
```

get_regions_by_name(*region_names=None*, *operator=None*, *attribute=None*)

Get all the relevant regions from the hierarchy based on the given parameters

Parameters

region_names:string or list fill the name or short name or list of names or short names of the regions in relations to operator parameter above. If no region names provided, the function will return all regions in the hierarchy.

operator [Operator] use the operator to find all the upper/lower level regions from a particular region name. For instance: Country > State (Country gt State).

Use the 'gt' operator to search for the upper level of State

attribute:string the region's attribute name that will be saved into the list (name, short_name, or col_name) if it's empty, then the list will store the object of the region

Returns

list list of regions or region's attribute. The function will return an empty list if there corresponding regions with the given name or short name are found.

Examples

```
>>> country = Region("Country")
>>> state = Region("State",col_name="STATE")
>>> sa4 = Region("Statistica Area 4",short_name="SA4",col_name="SA4")
>>> australia = GeoHierarchy(country,"Australia")
>>> australia.add_region(region=state, parent_region=country)
>>> australia.add_region(region=sa4, parent_region=state)
>>> regions = australia.get_regions_by_name()
>>> for region in regions:
>>>     print(region.name)
Country
State
Statistica Area 4
>>> regions = australia.get_regions_by_name(region_names='State', operator=le,)
>>> for region in regions:
>>>     print(region.name)
State
Statistica Area 4
>>> col_names = australia.get_regions_by_name(region_names=['State','SA4'],␣
↪attribute='col_name')
>>> for col_name in col_names:
>>>     print(col_name)
STATE
SA4
```

get_smallest_region_boundaries()

Get the smallest regional unit

Returns

Region the smallest regional unit

Examples

```
>>> country = Region("Country")
>>> state = Region("State",col_name="STATE")
>>> sa4 = Region("Statistica Area 4",short_name="SA4",col_name="SA4")
>>> australia = GeoHierarchy(country,"Australia")
>>> australia.add_region(region=state, parent_region=country)
>>> australia.add_region(region=sa4, parent_region=state)
>>> australia.get_smallest_region_boundaries().name
Statistica Area 4
```

property name

Return the name of the country

Returns

string The name of the country

Examples

The country's name can be set initially when calling the constructor. >>> australia = GeoHierarchy(country,"Australia") >>> australia.name 'Australia'

set_coordinate_boundary(*min_latitude*, *max_latitude*, *min_longitude*, *max_longitude*)

Set or modify the coordinate boundaries of a country.

Parameters

value:list The coordinate boundary of a country. The format of the input is : [minimum latitude, maximum latitude, minimum longitude, maximum longitude]

Examples

```
>>> australia = GeoHierarchy(country,"Australia")
>>> australia.set_coordinate_boundary(-43.58301104, -9.23000371,
                                      96.82159219, 167.99384663)
>>> australia.coordinate_boundary
[-43.58301104, -9.23000371, 96.82159219, 167.99384663]
```

property types

Return the dictionary that contains all the defined regional structures.

Returns

dictionary The defined regional structures

Examples

```
>>> country = Region("Country")
>>> state = Region("State",col_name="STATE")
>>> sa4 = Region("Statistica Area 4",short_name="SA4",col_name="SA4")
>>> australia = GeoHierarchy(country,"Australia")
>>> australia.add_region(region=state, parent_region=country)
>>> australia.add_region(region=sa4, parent_region=state)
>>> australia.add_type(sa4,"ASGS","Australian Statistical Geography Standard")
>>> australia.type
{'ASGS': 'Australian Statistical Geography Standard'}
```

2.2 Matcher

class addrmatcher.matcher.**DistanceMethod**(*value*)

Bases: enum.Enum

An enumeration.

JARO = 2

JARO_WINKLER = 3

LEVENSHTEIN = 1

class addrmatcher.matcher.**GeoMatcher**(*hierarchy*, *file_location=''*)

Bases: object

Methods

get_region_by_address(address[, ...])	perform address based matching and return the corresponding region e.g.
get_region_by_coordinates(lat, lon[, n, km, ...])	perform coordinate_based matching and return the corresponding regions in a dictionary e.g.

get_region_by_address(*address*, *similarity_threshold=0.9*, *nlargest=1*, *regions=None*, *operator=None*, *address_cleaning=False*, *method=DistanceMethod.LEVENSHTEIN*)

perform address based matching and return the corresponding region e.g. administrative level or statistical area

Parameters

address:string The complete physical address

similarity_threshold:float The minimum similarity ratio ranges between 0 and 1 (default = 0.9)

nlargest:int The number of the addresses to be returned by the function. If nlargest = 1, then the function will return the top similarity only (default = 1)

regions:string or list of string Specify the name or list of names of the regions to be returned by the function

operator: Operator use the operator (Operator.ge or Operator.le) to find all the upper/lower level regions from a particular region name. For instance: Country >= State (Country ge

State).

Use the 'ge' operator to search for the upper level of State (and itself)

address_cleaning:boolean whether to perform data cleansing on the address, for instance: revise invalid suburb name (currently, only applied to Australian addresses. Set this parameter as True for non-Australian addresses could return an error)

method:string The name of the edit distance algorithm used. Select one of DistanceMethod.LEVENSHTEIN,DistanceMethod.JARO, or DistanceMethod.JARO_WINKLER

Returns

Dictionary the dictionary of the matched adddresses. The keys of the dictionary are based on the column name defined in the Hierarchy object used. By default, the function will return only the top similarity record (nlargest = 1) as long as its similarity is larger than the threshold ratio. If no addresses have a similarity ratio more than the threshold, the function will return an empty dictionary.

Examples

```
>>> matcher = GeoMatcher(AUS)
>>> matched = matcher.get_region_by_address("2885 Darnley Street,
              Braybrookt, VIC 3019", similarity_threshold = 0.95)
>>> matched
{'MB_CODE_2016': ['20375120000'],
 'SA4_NAME_2016': ['Melbourne - West'],
 'SA3_NAME_2016': ['Maribyrnong'],
 'SA2_NAME_2016': ['Braybrook']
 'SA1_7DIGITCODE_2016': ['2134703'],
 'STATE': ['VIC'],
 'RATIO': [0.9841269841269842],
 'SSC_NAME_2016': ['Braybrook'],
 'LGA_NAME_2016': ['Maribyrnong (C)'],
 'FULL_ADDRESS': ['2885 DARNLEY STREET BRAYBROOK VIC 3019']}
```

get_region_by_coordinates(*lat*, *lon*, *n=1*, *km=1*, *regions=None*, *operator=None*)

perform coordinate_based matching and return the corresponding regions in a dictionary e.g. administrative level or statistical area

Parameters

lat:float latitude

lon:float longitude

n:integer the number of nearest addresses to be returned by the function.

km:integer the nearest addresses will be searched from the input coordinates point within the argument kilometer radius

Returns

Dictionary a dictionary of addresses with statistical and administrative regions. By default, the function will return the record with the smallest distance only (n = 1). If no addresses found within the radius (km), the function will return an empty dictionary.

Examples

```
>>> matcher = GeoMatcher(AUS)
>>> matched = matcher.get_region_by_address(-26.657299,153.094955)
>>> matched
{'FULL_ADDRESS': ['8 32 SECOND AVENUE MAROOCHYDORE QLD 4558'],
 'LATITUDE': [-26.6572865955204],
 'LONGITUDE': [153.09496396875],
 'LGA_NAME_2016': ['Sunshine Coast (R)'],
 'SSC_NAME_2016': ['Maroochydore'],
 'SA4_NAME_2016': ['Sunshine Coast'],
 'SA3_NAME_2016': ['Maroochy'],
 'SA2_NAME_2016': ['Maroochydore - Kuluin'],
 'SA1_7DIGITCODE_2016': ['3142707'],
 'MB_CODE_2016': ['30563074700'],
 'DISTANCE': [0.0016422183328786543]}
```

2.3 Region

Data structure for the regional unit

class addrmatcher.region.**Region**(*name: str*, *short_name: str = ''*, *col_name: str = ''*)

Bases: object

The Region class represents a unit of area in the country. Each region has a unique name and a corresponding column in the reference dataset.

Parameters

name: string The name of the area unit

short_name: string The short name of the area unit

col_name: string The column name of the area unit

Examples

The area's column name can be set initially when calling the constructor. >>> sa2 = Region('Statistical Area 2',short_name='SA2',col_name='SA2') >>> sa2.col_name 'SA2'

col_name: str = ''

name: str

short_name: str = ''

2.4 Resource

addrmatcher.resource.**create_url**(*url*)

Produce a URL that is compatible with Github's REST API from the input url.This can handle blob or tree paths.

Parameters

url [str] url to the data directory in Github repository

Returns

str Github API url

str Download directory

addrmatcher.resource.**download**()

Trigger the download_data function and read the argument from user's command line interface.

addrmatcher.resource.**download_data**(*country='Australia'*, *output_dir='/home/docs/checkouts/readthedocs.org/user_builds/addrmatcher/checkouts/lates*

Download the files in directories and sub-directories in repo_url.

Parameters

country [str] country name which will be sub-directory name example - data/Australia/.

Returns

int number of total files downloaded

addrmatcher.resource.**print_text**(*text: str*, *color: str = 'default'*, *in_place: bool = False*, ***kwargs: any*) → None

Print text to console.

Parameters

text [str] text to print

color [str] it can be one of "red" or "green", or "default"

in_place [bool] whether to erase previous line and print in place

****kwargs** [dict, optional] : other keywords passed to built-in print

CHAPTER

THREE

MODULES

- modindex

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

a

INDEX