Parsing Challenge

Minyi Liang
Minyi_liang@hotmail.com

Libraries selection:

In this task, I used 3 libraries to parse dates (Dateutil.parser, Dateparser and Datefinder) and 1 library to parse addresses (postal.parser). The main reasons to select these libraries are provided and some of their advantages/disavantages are discussed as follows.

1. Dateutil.parser
This is one of the most popular libraries to parse dates in Python, it was designed not only for formatted dates, e.g. '22-10-2015', but also natural language dates such as 'Jan 2011'. However, **it does not support multi-languages**.

2. Dateparser
Dateparser is built on top of Dateutil.parser library, the main advantage is that it has automatic language detection feature.

However, **neither library 1 nor 2 can extract date information from texts**. For example, it does not recognise dates in string 'New York, Jan 2011'.

3. Datefinder
Datefinder is a python module for locating dates inside text. It can extract all sorts of date-like strings from a document and uses the Dateparser package to turn them into datetime objects.

However, **it does not recognise dates mixed with time zones**, e.g., 2011/01/01 0000UTC.

4. Postal.parser***
Libpostal is a C library (with Python bindings: Postal.parser) for parsing/normalising street addresses around the world using statistical NLP and open data, generating sub-building components like apartment numbers and cities. It understands location-based strings in multiple languages.

However, it **works best with clean location data**, and it is not very robust to messy real-world input.

For this task, it is better to extract all the datetime strings from the texts before using Postal.parser to parse addresses.

Coding idea (**see also attached flowchart**):

1. General idea:
First, each string in the input JSON file is passed to library 1 (Dateutil.parser) to identify pure datetime strings (e.g., '22-10-2015'). If it returns errors, it is either the string is written in non-English or the string contains a mix of addresses and dates.

The string is then passed to library 2 (Dateparser) to indentify non-English dates.

If library 2 (Dateparser) returns errors, it means the string contains addresses.

It will then be passed to library 3 (Datefinder) to extract dates from the text. The remaining text containing only addresses will be passed to library 4 (Postal.parser) to parse addresses.

If the input string contains only addresses (library 3 returns empty list), it will be passed to library 4 (Postal.parser) directly to parse addresses.

2. Parsing incomplete dates
None of the above libraries work well with incomplete dates (e.g., '2011-01'). Library 1 (Dateutil.parser) allows users to set non-zero default values when parsing incomplete dates, for example, '2011-01' will be parsed as '2011-01-01' if the default day value is set to be '01'.

Utilising this feature, a function to parse incomplete dates is introduced:

1. Detect if input datetime string is incomplete (default values are used when parsing incomplete dates).
2. Use a different set of default values, if output parsed day/month/year changes as well, it means the day/month/year is not provided in the input datetime string.
3. Replaced default values with '0'.

For example, if we set default values as (1, 1, 1), then input '2011-01' will be parsed as '2011-01-01'. We then use a different set of default values (2, 2, 2), and input '2011-01' will be parsed as '2011-01-02'. Thus, we know that no day information is provided in the input string, and we replace the day with '0', and output a new parsed date as '2011-01-00'.

Ranking:

Then ranking is a 4-digit number, indicating whether the output contains an address, year, month, day, respectively. For example, it the output only contains the year, the ranking is "0100".

Evaluation:

To evaluate the output quality, I chose a small sample (e.g., 500 strings) from the input JSON file, and manual labelled all the roads/cities/countries/dates, and then compare the results of my model with the manual labelling. For example, each input string is labelled as X(i), the resulting parsed string from my model is labelled as Y_model(i), and the real value is labelled as Y(i). Then I measure the false positive and false negative cases.

I have also compared the results from my model with those of other algorithms or libraries (e.g., GeoPy geocoder, Geograpy etc.).

Improvement:

The 3 libraries for date parsing are built on top of each other, and therefore have repeated functions. For example, library 2 (dateparser) is vey similar to library 1 (dateutil.parser), but it has the advantage of supporting multiple languages; library 3 (datefinder) is, again, very similar to library 1&2, but it can extract datetime information from fuzzy texts. If more time is allowed, one should combine the 3 libraries and build a stand-alone library that supports multiple languages as well as entity recognition to improve efficiency.

The function I introduced to parse incomplete dates is not very efficient. An easy improvement would be to change the source code of library 1 (dateutil.parser) to allow (0, 0, 0) default values. At the moment, library 1 only supports non-zero default values.

Library 4 (postal.parser) is not very robust to real world fuzzy strings, it considers all input strings as addresses. One can use another library (for example, Spacy) to recognise entities first, and then pass only the location information to library 4 (postal.parser) to parse addresses.

*** How to install postal.parser (the following commands apply to Mac OS only)

1. Install homebrew (if you don't have one)
```
>> /usr/bin/ruby -e "$(curl –fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

2. Install the libpostal C library
```
>> brew install curl autoconf automake libtool pkg-config
```

3. Install libpostal
```
>> git clone https://github.com/openvenues/libpostal
>> cd libpostal
>> ./bootstrap.sh
>> ./configure --datadir="xxxxx Any given directory xxxxx"
>> make
>> sudo make install
```

4. Install python library
```
>> pip install postal
```