COMP6714 (16S2) PROJECT

DUE DATE: 23:59 10 OCT 2016 (MON)

1. Objective

In this project, you will build a relation extractor to extract specific relations from a given corpus.

Note that it will take you quite some time to complete this project even if you are familiar with various Python libraries and have good Python programming experience. Therefore, we earnestly recommend that *you start working on this project as early as possible*.

2. Background

Text documents often contain valuable structured data that is hidden in regular English sentences. To extract them, we usually takes two steps. The first step is to perform Named Entity Recognition (NER) to identity entity mentions and classify them into correct types (`PERSON`, `ORG`, `DATE`, `GPE`[1], etc.). The second step is to perform relation extraction exploiting NER results.

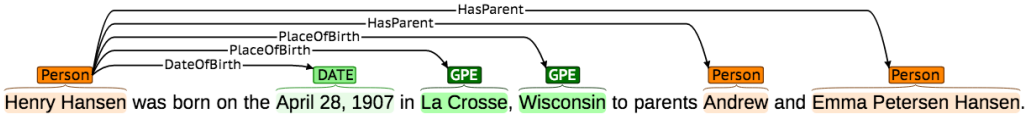

Figure 1. Example Sentence

For example, in the sentence in Figure 1, there are 6 entities — 3 `PERSON` entities, 1 `DATE` entity, 2 `GPE` entities. There are 5 different relations that hold between those 6 entities. In this project, the NER results are given, and you will focus on the Relation Extraction step.

Relations are usually represented as triplets of (*subject*, *predicate*, *object*). For example, the above 5 relations can be represented as:

```
("Henry Hansen", DateOfBirth, "April 28,1907")
("Henry Hansen", PlaceOfBrith, "La Crosse")
("Henry Hansen", PlaceOfBrith, "Wisconsin")
("Henry Hansen", HasParent, "Andrew")
("Henry Hansen", HasParent, "Emma Petersen Hansen")
```

[1] `GPE` stands for geo-political entities such as city, state/province, and country.

3. Overview of the Tasks

You need to implement a program in Python 3 that extracts two specific relations from an input file consisting of sentences and NER results. The two relations are : `DateOfBirth`, and `HasParent`.

You also need to submit a report (in the PDF format) which answers the following questions:

- A detailed description of your approach, why you chose such a approach and how you formulated the task.
- A detailed description of the pattern used in your extractor. How do you discover those pattern.
- How do you experiment and improve your extractor?

4. Entity Types

All input sentences have already been annotated with external tools and manually cleaned. We list some common entity types in Table 1. For the full list, please refer to `https://spacy.io/docs/`.

Table 1. Entity Types

Entity Type	**Description**
`PERSON`	People, including fictional.
`DATE`	Absolute or relative dates or periods.
`GPE`	Countries, cities, states.

5. Relation Types

Among those 3 entity types, many relation types can be defined. In Table 2, we list several relations that can be define between entities. In this project, we only evaluate two specific relations though (See Section 3).

6. Training Data and Format

We will release a training dataset consisting of sentences that contain at least one instance of the two relations to be extracted. All training data are contained in one single `json` file named `training.json`. This `json` file is a list of sentence objects. Each sentence object is wrapped by a dictionary. The `sentence_id` field is used to identify the sentence. The `sentence` field is a dictionary that contains the target sentence and pre-filled annotations. `text` is the sentence text. `annotation` is a list of tuples that contains pre-filled annotations for the sentence. Each tuple represent a token in the sentence and contains 5 fields. The 1st field is the order of the token in the sentence. The 2nd field is the word of this token in the sentence. The 3rd field is the token after lemmatization. The 4th field

TABLE 2. Relation Types

Relation Type	**Subject Type**	**Object Type**	**Description**	Eval.
`DateOfBirth`	`PERSON`	`DATE`	The date on which the assigned person was born.	Yes
`HasParent`	`PERSON`	`PERSON`	The parents of the person. In addition to biological parents, step-parents and adoptive parents are also acceptable.	Yes
`PlaceOfBirth`	`PERSON`	`GPE`	The Place in which the assigned person was born.	No
`AlternateName`	`PERSON`	`PERSON`	Names used to refer to the person that are distinct from the "official" name. Alternate names may include aliases, stage names, alternate transliterations, abbreviations, alternate spellings, nicknames, or birth names.	No

is the Part-Of-Speech tag of the token in this sentence. The 5th field is the Named Entity Tag in IOB format[2].

The positive ground truth relations is contained in the `relations` field. The `relations` field is a list of relation triplets. Each triplets contains "subject", "predicate", and "object" fields which are the main target of the output.

```
[
{
  "sentence_id":"TR.00001",
  "sentence":{
    "text":"Bill was born 1986."
    "annotation":{
      (1, "Bill", "bill", "NNP", "B-PERSON"),
      (2, "was", "be", "VBD", "O"),
      (3, "born", "bear", "VBN", "O"),
      (4, "1986", "bill", "CD", "B-DATE"),
      (5, ".", ".", ".", "O"),
    }
  },
  "relations":[
  {
    "subject":"Bill",
    "predicate":"DateOfBirth",
    "object":"1986"
  },
  ]
```

[2] `https://en.wikipedia.org/wiki/Inside_Outside_Beginning`

```
},
...
]
```

Listing 1. Example Input Data

7. Code Skeleton and Output

In the project website, we will release a tar file named `proj6174.tar`. `proj6714.tar` will contain 5 files.

- `training.json` : This is the training data.
- `config.py` : This is the file that contains the environment settings. You can modify it based your own environment. **DO NOT ADD NEW VARIABLES**.
- `relation.py`: Defines the relation types. **DO NOT MODIFY THIS FILE**.
- `run.py`: This file is the entry point of execution. It reads in the data, calls the extraction code, and evaluate the results of the extraction by reporting the F1 score. **DO NOT MODIFY THIS FILE**.
- `extractor.py`: This file is the starting point of your own code. It contains two functions, `extract_date_of_birth` and `extract_has_parent`, that will be called to perform the extraction. **DO NOT REMOVE THOSE TWO FUNCTIONS.**

In the code below, we show the skeleton code of `extractor.py`, this source code defined the interface. The input argument `sentence` is the `sentence` data from input data. You can only replace the middle section by your own code but you **cannot** change the interfaces. We will only call these two functions to evaluate your submission. The output of those functions is a list of `Relation` objects. You can read the code of `relation.py` to understand the data structure.

```
# -*- coding: utf-8 -*-

import config
from relation import Relation

def extract_date_of_birth(sentence):
    predicate = "DataOfBirth"
    results = []

    ########################################################################
    # Replace this part to your own code of extracting DataOfBirth.
    ########################################################################

    return results

def extract_has_parent(sentence):
    predicate = "HasParent"
    results = []

```

```
20      ###################################################################
21      # Replace this part to your own code of extracting HasParent.
22      ###################################################################
23
24      return results
```

Listing 2. extractor.py

8. External Library

The project encourages you to design you own pattern-based relation extractor. We do not allow you to use external library that can perform relation extraction straightaway. The testing environment is based on a standard Anaconda 4.1.1 and Python 3.5 installation. There are some external libraries available to you:

- NLTK, version 3.2.1.
- Spacy, version 0.101.0.
- Stanford Core NLP, 3.6.0. The libraries can be accessed through NLTK. The path variable is located in the `config.py` file.

You can run the following code in the skeleton to test if your environment has been configured correctly or not.

```
python run.py training.json
```

If you want to use other external libraries for this project, please send email to LiC and give strong reasong for needing it. We will evaluate each request and reply you with the decision. If a library is approved, we will update them in the online submission system and make it known to all students in the class.

9. Submission

You need to use our online submission system[3] to submit your code. You will need to login before submitting your code or viewing your results. The id is your student number (e.g., `z1234567`) and your password will be sent to you by email.

To submit, please `tar` your source codes into a file named `proj1.tar.gz`[4]. This file size is limited to 5MB.

Your submission should contain at least one files named

- `extractor.py`

in the root directory. You are free to add any other Python files. But files named `run.py` and `relation.py` will be replaced with our standard version. DO NOT PUT YOUR CODE in these two files.

You need to use our online submission system `http://kg.cse.unsw.edu.au:8714/` to submit your report as well. We only keep the last version for the evaluation.

[3]The URL will be sent to you by email.

[4]Use the command: `tar cvfz proj1.tar.gz file1 file2 ...`

10. Evaluation

Program Evaluation. The online submission system will unpack your submission and run your code on a testing datasets. Your code should complete the processing within **10 minutes**. Your results will be revealed online within 24 hours. You are limited to submit at most 10 times. For each submission, the system will output the scores of each relation and the mark of the this submission based on the marking scheme. Your final score will be based on the one with the highest mark achieved.

In this project we will focusing only on two relations, one is `DateOfBirth` and another is `HasParent`. Their score are marked separately and each has maximimum 40 marks. The rest 20 marks is based on your report.

Evaluation Measure. The output of the extractor is a list of objects of `Relation`s defined in `relation.py`. Each Relation object contains three attributes, *subject*, *predicate*, and *object*. When measure for correctness, we will check all three values of your output with the correct relation instances extracted from the input. Only when all three attributes are matched, the results is regarded as a correct one. The comparison is case-insensitive. Please refer to the function `check_correct` in the `run.py` file for details.

Note: for sentences that contains multiple named entities that refer to the same person, we only use the left-most one as the entity for the relation. For example:

```
Williams was born Robin McLaurin Williams at St.  Luke's Hospital
in Chicago, Illinois on July 21, 1951
```

Both `Williams` and `Robin McLaurin Williams` refer to the same person. We regard the following triplet:

```
("Williams", DateOfBrith, "July 21, 1951").
```

correct, but this one:

```
("Robin McLaurin Williams", DateOfBrith, "July 21, 1951").
```

incorrect.

Given the output of the extractor, the system will first evaluate the precision ($precision = \frac{\#Correct}{\#Output}$) and recall ($recall = \frac{\#Correct}{\#GroundTruth}$)[5]. Then the final score is the $F1$ measure of both precision and recall

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

[5] $\#GroundTruth$ is the number of relations instances of a specific type in the input data.

To convert the $F1$ score (0 to 1) measure to $Mark$ (0 to 100) for each relation extractor, we use a piecewise linear function:

$$Mark(x) = \begin{cases} 0 & x < a \\ 60 \cdot \frac{x-a}{b-a} & a \le x < b \\ 60 + 20 \cdot \frac{x-b}{c-b} & b \le x < c \\ 80 + 20 \cdot \frac{x-c}{d-c} & c \le x < d \\ 100 & d < x \end{cases}$$

There are four parameters in the above formula: a, b, c, and d (also called *check points*). For example, if you want to obtain at least 60 marks, your $F1$ score must be no less than a. Each relation will have a separate set of check point values. We will release the value in our online submission system.

Report Submission. Make sure you include your name and student ID in the report.

Bouns. For the extractor that reaches a F1-score that exceeds the check point d, Your extractor will be entered into a "contest". If it belongs to the 10 best performing submissions, you will get at most 20 bouns points. More specifically, the 1st place will get 20 points, the 2nd place will get 18 points, and so on and so forth.

Late Panelty. The online submission system will be closed 5 days past the due day. -10% per day for each of the first 2 days, and -20% per day for each of the following days.

Plagiarism. Make sure you read "8. Academic honesty and plagiarism" in `http://www.cse.unsw.edu.au/~cs6714/16s2/intro.html`