TM-L2

November 13, 2017

1 L2: Information Extraction

In this lab you will implement and evaluate a simple system for information extraction. The task of the system is to read sentences and extract entity pairs of the form *x*–*y* where *x* is a person, *y* is an organisation, and *x* is the 'leader' of *y*. Consider the following example sentence:
Mr. Obama also selected Lisa Jackson to head the Environmental Protection Agency.
From this sentence the system should extract the pair

```
("Lisa Jackson", "Environmental Protection Agency")
```

The system will have to solve the following sub-tasks: * entity extraction – identifying mentions of person entities in text * relation extraction – identifying instances of the 'is-leader-of' relation

The data set for the lab consists of 62,010 sentences from the Groningen Meaning Bank (release 2.2.0), an open corpus of English. To analyse the sentences you will use spaCy.

1.1 Getting started

The first cell imports the Python module required for this lab.

```
In [ ]: import tm2
```

The data is contained in the following file:

```
In [ ]: data_file = "/home/TDDE16/labs/l2/data/gmb.txt"
```

The `tm2` module defines a function `read_data` that returns an iterator over the lines in a file. You should use this function to read the data for this lab. Use the optional argument `n` to restrict the iteration to the first few lines of the file. Here is an example:

```
In [ ]: for sentence in tm2.read_data(data_file, n=3):
            print(sentence)
```

The next cell imports spaCy and loads its English language model.

```
In [ ]: import spacy

        nlp = spacy.load('en', disable=['textcat'])
```

1.2 Entity extraction

To implement the entity extraction part of your system, you do not need to do much, as you can use the full natural language processing power built into spaCy. The following code extracts the entities from the first 5 sentences of the data:

```
In [ ]: for i, doc in enumerate(nlp.pipe(tm2.read_data(data_file, n=5))):
            for ent in doc.ents:
                print("{}\t{}\t{}\t{}".format(ent.text, ent.start, ent.end, ent.label_))
```

Read the section about named entities from spaCy's documentation to get some background on this. (Please note that we are using version 1 of the spaCy library, which means that there may be slight differences in the usage. At the time of writing, the current version 2 is not yet stable and fast enough for this lab.)

1.3 Problem 1: Extract relevant pairs

The first problem that you will have to solve is to identify pairs of entities that are in the 'is-leader-of' relation, as in the example above. There are many ways to do this, but for this lab it suffices to implement the strategy outlined in the section on Relation Extraction in the book by Bird, Klein, and Loper (2009):

- look for all triples of the form (X, α, Y) where X and Y are named entities of type *person* and α is the intervening text
- write a regular expression to match just those instances of α that express the 'is-leader-of' relation

You can restrict your attention to adjacent pairs of entities – that is, cases where X precedes Y and α does not contain other named entities.

Write a function `extract` that takes an analysed sentence (represented as a spaCy `Doc` object) and yields pairs (X, Y) of strings representing entity mentions predicted to be in the 'is-leader-of' relation.

```
In [ ]: def extract(doc):
            """Extract relevant relation instances from the specified document.

            Args:
                doc: The sentence as analysed by spaCy.
            Yields:
                Pairs of strings representing the extracted relation instances.
            """
            # TODO: Replace the following line with your own code
            return tm2.extract(doc)
```

The following cell shows how your function is supposed to be used. The code prints out the extracted pairs for the first 1,000 sentences in the data. It additionally numbers each pair with the identifier of the sentence (line number in the data file) which it was extracted from. Note that the sentence (line) numbering starts at index 0.

```
In [ ]: for i, doc in enumerate(nlp.pipe(tm2.read_data(data_file, n=1000))):
            for person, org in extract(doc):
                print("{}\t{}\t{}".format(i, person, org))
```

Once you feel confident that your `extract` function does what it is supposed to do, execute the following cell to extract the entities from the full data set. Note that this will probably take a few minutes.

```
In [ ]: extracted = set()
        for i, doc in enumerate(nlp.pipe(tm2.read_data(data_file))):
            for person, org in extract(doc):
                extracted.add((i, person, org))
```

After executing the above cell, all extracted id-string-string triples are in the set `extracted`. The code in the next cell will print the first 10 triples in this set.

```
In [ ]: for i, person, org in sorted(extracted)[:10]:
            print("{}\t{}\t{}".format(i, person, org))
```

1.4 Problem 2: Evaluate your system

You now have an extractor, but how good is it? To help you answer this question, we provide you with a 'gold standard' of entity pairs that your system should be able to extract. The following code loads them (again augmented with the relevant sentence id) from the file `gold.txt` and adds them to the set `gold`:

```
In [ ]: gold_file = "/home/TDDE16/labs/l2/data/gold.txt"

        gold = set()
        with open(gold_file) as fp:
            for line in fp:
                columns = line.rstrip().split('\t')
                gold.add((int(columns[0]), columns[1], columns[2]))
```

The following code prints the 10 first pairs from the gold standard:

```
In [ ]: for i, person, org in sorted(gold)[:10]:
            print("{}\t{}\t{}".format(i, person, org))
```

Your task now is to write code that computes the precision, recall, and F1 measure of your extractor relative to the gold standard.

```
In [ ]: def evaluate(reference, predicted):
            """Print out the precision, recall, and F1 for the id-entity-entity
            triples in the set `predicted`, given the triples in the reference set.

            Args:
                reference: The reference set of triples.
                predicted: The set of predicted triples.
```