CE807 Lab 2
Named Entity Recognition & Technical Writing

In this lab we are going to experiment with Named Entity Resolution (NER), first using Python libraries, then using tools outside Python, in particular CRF++.

1. Off-the-shelf NE Recognizers: The Stanford NER

There are several publicly available NER systems, some of them available as part of complete pipelines.

One of the most widely used such NER systems is the Stanford NER system, part of the Stanford CORENLP information extraction pipeline available from (and described at)

http://stanfordnlp.github.io/CoreNLP/

Stanford CoreNLP includes a POS tagger, parser, NER system. coreference resolver, and sentiment analyser, and supports Arabic, Chinese, English, French, German and Spanish

ToDo: read the description of the Stanford CoreNLP pipeline at the page above.

The Stanford NER system is described at

http://stanfordnlp.github.io/CoreNLP/ner.html

like most modern NER systems, is based on the Conditional Random Fields sequence learning model. It is written in Java and supports English, Chinese, and German.

ToDo: read the description of the Stanford NER system at the page above

The system can be downloaded from a number of sites, including the Github site above and from

http://nlp.stanford.edu/software/CRF-NER.shtml

Three different pre-trained models are included with the package:

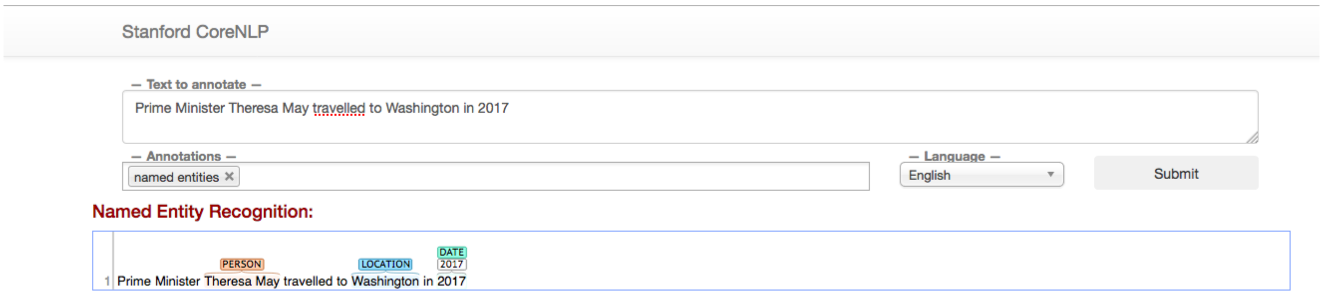
1. **english.all.3class.distsim.crf.ser.gz**: a 3 class NER tagger that can label Person, Organization, and Location entities. It is trained on data from CoNLL, MUC6, MUC7, and ACE.

2. **english.conll.4class.caseless.distsim.crf.ser.gz**: a 4 class NER tagger that can label Person, Organization, Location, and Misc. It is trained on the CoNLL 2003 Shared Task.
3. **english.muc.7class.caseless.distsim.crf.ser.gz**: It is trained o from MUC and distinguishes between 7 different classes

One of these models has to be selected when using the Stanford NER parser, either from the command line or from within Python / NLTK (see below).

There is a demo of the system at:

http://corenlp.run/
(To see only NER, you can unselect the other types of annotations). The output of the demo is shown in the following picture.


Stanford CoreNLP

Text to annotate: Prime Minister Theresa May travelled to Washington in 2017

Annotations: named entities

Language: English

Named Entity Recognition:

1 Prime Minister Theresa May travelled to Washington in 2017

ToDo: experiment with the demo, trying to break it. Try first in English, then try other languages.

In the next Section, we will see one way for using the Stanford NER from Python using the NLTK interface.

2. NER in NLTK

The NLTK Python library includes a number of NE taggers and interfaces to several others, including the Stanford tagger.

2.1 The standard NLTK NE tagger

The standard NER in NTK is the function `ne_chunk()`. The example in the script `ne_chunk_ex.py` shows how to use it:

```
import nltk
import re
import time

exampleArray = ['Prime Minister Theresa May travelled to
Washington in 2017.']

##let the fun begin!##
def nerInNLTK():
    try:
        for item in exampleArray:
            tokenized = nltk.word_tokenize(item)
            pos_tagged = nltk.pos_tag(tokenized)
            print (pos_tagged)

            namedEnt = nltk.ne_chunk(pos_tagged)
            namedEnt.draw()

            time.sleep(1)

    except Exception as e:
        print(e)

nerInNLTK()
```

Note: this example uses Python 3 syntax. Also: in order to run it you'll need to have the resources `punkt` and `averaged_perceptron_tagger` installed. If they are not installed, run `nltk.download()` and look under Models.

Or install them directly using:

```
>>> import nltk
>>> nltk.download('maxent_ne_chunker')
>>> nltk.download('words')
...
```

ToDo: try different examples. Make sure you understand the syntax of the output.

2.2 The Stanford Tagger in NLTK

An interface to the Stanford taggers is included in NLTK as part of the `nltk.tag.stanford` module, described at:

http://www.nltk.org/api/nltk.tag.html#module-nltk.tag.stanford

In particular, the Stanford NERTagger can be used as by invoking the function `nltk.tag.stanford.StanfordNERTagger()`, that requires as an obligatory first argument the name of the model to be used. The second argument is the location of the NER executable, `stanford-ner.jar`—you can either specify that in the CLASSPATH variable or specify it explicitly (which is what we will do). In order to run the tagger, you need (i) to make sure that Java is installed, (ii) to download the appropriate models and the archive `stanford-ner.jar` either from this lab's page or by downloading a version of the Stanford NER system from

http://nlp.stanford.edu/software/CRF-NER.shtml#Download

The latest version of the package (version 3.7) requires Java 8. To run on a machine with Java 6 or 7, you need version 3.4.1. In the module page you can find files for both versions.

Put the files in the folder you are using as working directory in Python:
- the model `english.all.3class.distsim.crf.ser.gz` (3 classes: Person, Organization, and Location) and
- the archive `stanford-ner.jar`
and then try to get the following code to run (script stanford_NER_NLTK.py)

```
from nltk.tag.stanford import StanfordNERTagger

st = StanfordNERTagger('english.all.3class.distsim.crf.ser.gz', './stanford-ner.jar')

st.tag('Prime Minister Theresa May travelled to Washington in 2017'.split())
```

ToDo: try to replace the model used in this script with a different model from those distributed with the package.

3. Writing Technical Reports on Data Analyses such as your Assignment Report

A very important part of conducting data analyses on any kind of data is to read corresponding literature and writing reports on the findings from your analysis. Thus, we devote the remaining time of the lab to understand a bit more the rules and challenges of writing technical reports. This is required for both assignments.

- *Literature search:* In a first step, please download the first assignment and read it (if you yet have not done so). Besides the text analysis, there is a report to be produced.

Writing a technical report follows very standardised and established rules. It involves finding literature relevant to the text analysis task and contextualising this related work, i.e. putting the literature you found into context to the given task. In a subsequent step, the features and methods you want to apply need to be defined. After training the model on these features using your method(s) of choice, the results need to be documented. A technical report concludes with a good discussion of the results, i.e. reflection of what you have observed. Furthermore, it is common practise to also reflect on lessons learned, i. e. to discuss what worked well and what not, what you would do differently if asked to do the task again etc., before you conclude.

Thus, the following steps are:

- *Define a ToC:* Outline a generic structure of a technical document to report a text analysis task, i. e. prepare a general Table of Content (ToC). A good approach here is to learn and abstract from the existing research papers like those identified in the literature search above.

- *Adapt to a specific problem*: Apply and specialise your ToC to the task of your first assignment. What sections need to be refined, which ones can stay generic?

A classical document given to many students writing technical reports are the renown 2006 “Tips for Writing Technical Papers” by Jenifer Widom. You find the document included as PDF in the Lab or online at: https://cs.stanford.edu/people/widom/paper-writing.html

One can learn a lot from experienced researchers liker Jennifer Widom, who is a distinguished expert in database systems. We try to imbibe from her experience.

- Thus, as next step please go through the document "Tips for Writing Technical Papers" by Jenifer Widom. Reflect on her tips for technical paper writing to your proposal for the assigment.

- As last step, finalise the ToC and first draft of your report for your first assignment. You can reuse the ToC later, when you are working on the second assignment.

Please note, the technical writing skills and steps conducted above are essential for any future job as Big Data Scientist both in industry as well as academia. Specifically, it will help you a lot when you are writing your Master's thesis since essentially the same rules and structures apply.

Good luck!