Introduction to Human Language Technologies
Lab.7: NERC

Natural Language Research Group

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Course 2018/19

Outline

Session requirements (I)

Maximum Entropy Name Entity Chunker:

- Both Linux and Windows (via `python`)

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

Stanford CoreNLP:

- Linux (via shell)
  - Install CoreNLP: `https://stanfordnlp.github.io/CoreNLP/download.html`
  - and execute CoreNLP server:

```
> java -mx4g -cp "whole_path/stanford-corenlp-full-2017-06-09/*"
edu.stanford.nlp.pipeline.StanfordCoreNLPServer
-port 9000 -timeout 15000
```

  - make sure you have intalled the later java version

Outline

NERC models in NLTK

Default model

- maximum entropy model (PERSON, LOCATION, ORGZATION) trained with ACE corpus

```
from nltk import ne_chunk
ne_chunk(POS_tagged_sent, [binary=True/False])
```

- `binary`: True is used just to recognize NEs, without classifying them into the three NE classes. By default, it is set to False
- Output: tree format by default. Use `nltk.chunk.tree2conllstr(ner_output)` to get CoNLL format

- NLTK doesn't have an English corpus for NERC (CoNLL 2002 corpus for Spanish and Dutch)

NERC example

Example:

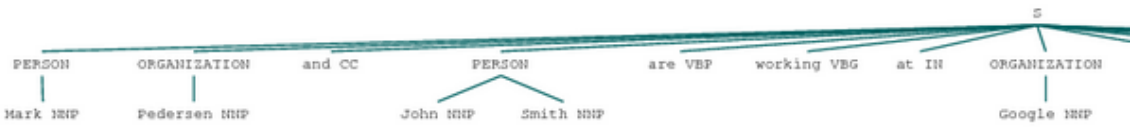
In [1]:
```
from nltk import word_tokenize, pos_tag, ne_chunk

sentence = "Mark Pedersen and John Smith are working at Google " + \
           "since 1994 for $1000 per week."

x = pos_tag(word_tokenize(sentence))
res = ne_chunk(x)
print(res)
res
```

```
(S
  (PERSON Mark/NNP)
  (ORGANIZATION Pedersen/NNP)
  and/CC
  (PERSON John/NNP Smith/NNP)
  are/VBP
  working/VBG
  at/IN
  (ORGANIZATION Google/NNP)
  since/IN
  1994/CD
  for/IN
  $/$
  1000/CD
  per/IN
  week/NN
  ./.)
```

Out[1]:



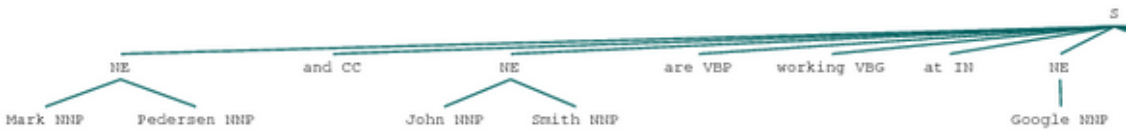
NERC example

Example:

In [2]:
```
res = ne_chunk(x, binary=True)
print(res)
res
```

```
(S
  (NE Mark/NNP Pedersen/NNP)
  and/CC
  (NE John/NNP Smith/NNP)
  are/VBP
  working/VBG
  at/IN
  (NE Google/NNP)
  since/IN
  1994/CD
  for/IN
  $/$
  1000/CD
  per/IN
  week/NN
  ./.)
```

Out[2]:



NERC models in CoreNLP

Third party model: Stanford CoreNLP server

- CRFs and rule models (PER, LOC, ORG, DATE, TIME, MONEY, ...)

```
from nltk.tag.stanford import CoreNLPNERTagger
CoreNLPNERTagger(url='http://localhost:9000').
tag(tokenized_sent)
```

  make sure you have installed at least nltk version 3.2.5

- Previously, install CoreNLP:

  `https://stanfordnlp.github.io/CoreNLP/download.html`

- and execute CoreNLP server:

```
java -mx4g -cp
"whole_path/stanford-corenlp-full-2017-06-09/*"
edu.stanford.nlp.pipeline.StanfordCoreNLPServer -port 9000
-timeout 15000
```

  make sure you have intalled the later java version

Stanford CoreNLP

Example:

In [3]:
```
from nltk.tag.stanford import CoreNLPNERTagger

CoreNLPNERTagger(url='http://localhost:9000').tag(word_tokenize(sentence))
```

Out[3]:
```
[('Mark', 'PERSON'),
 ('Pedersen', 'PERSON'),
 ('and', 'O'),
 ('John', 'PERSON'),
 ('Smith', 'PERSON'),
 ('are', 'O'),
 ('working', 'O'),
 ('at', 'O'),
 ('Google', 'ORGANIZATION'),
 ('since', 'O'),
 ('1994', 'DATE'),
 ('for', 'O'),
 ('$', 'MONEY'),
 ('1000', 'MONEY'),
 ('per', 'O'),
 ('week', 'DURATION'),
 ('.', 'O')]
```

Mandatory exercise

Statement:

1. Read all pairs of sentences of the trial set within the evaluation framework of the project.
2. Compute their similarities by considering the following approaches:
   - words and Jaccard coefficient (same as in Session 5)
     ex: `words=['John','Smith','is','working']`
   - words plus NEs and Jaccard coefficient
     ex: `word_and_NEs=['John Smith','is','working']`
3. Show the results. Do you think it could be relevant to use NEs to compute the similarity between two sentences? Justify the answer.

Outline

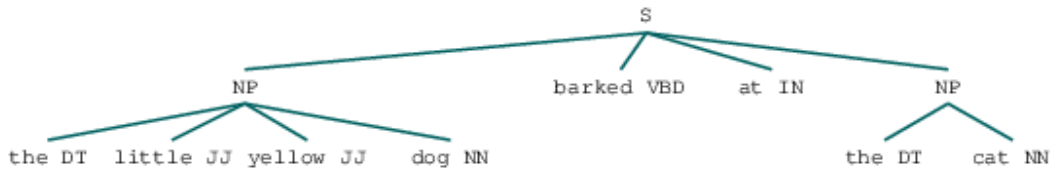
NLTK RegexpParser

In [1]:

```
1  import nltk
2
3  sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"),("dog", "NN"),\
4              ("barked", "VBD"), ("at", "IN"), ("the", "DT"), ("cat", "NN")]
5
6  grammar = "NP: {<DT>?<JJ>*<NN>}"
7
8  cp = nltk.RegexpParser(grammar)
9  result = cp.parse(sentence)
10
11 print(result)
12 result
```

```
(S
  (NP the/DT little/JJ yellow/JJ dog/NN)
  barked/VBD
  at/IN
  (NP the/DT cat/NN))
```

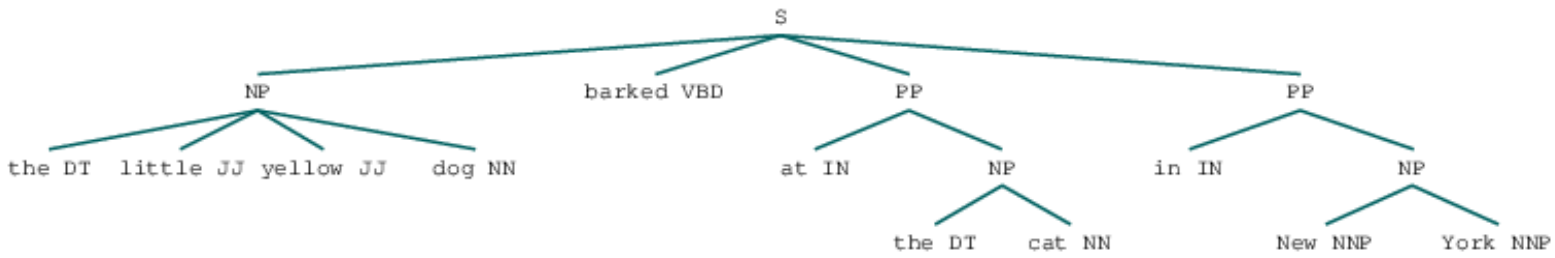
Out[1]:



Optional exercise

Enlarge the previous grammar to parse the next sentence as follows:

```
[("the", "DT"), ("little", "JJ"), ("yellow", "JJ"),("dog", "NN"),\
 ("barked", "VBD"), ("at", "IN"), ("the", "DT"), ("cat", "NN"), \
 ("in", "IN"), ("New", "NNP"), ("York", "NNP")]
```



Conll 2000 Corpus

Chunking evaluation:

In [3]:
```
1 from nltk.corpus import conll2000
2 test_sents = conll2000.chunked_sents('test.txt', chunk_types=['NP'])
3 print(cp.evaluate(test_sents))
```

```
ChunkParse score:
    IOB Accuracy:  59.7%%
    Precision:     45.3%%
    Recall:        24.2%%
    F-Measure:     31.6%%
```

Example:

These two sentences have same PoS labels an different chunks.

a. `Joey/NN sold/VBD the/DT farmer/NN rice/NN ./.`

b. `Nick/NN broke/VBD my/DT computer/NN monitor/NN ./.`

Conll 2000 Corpus

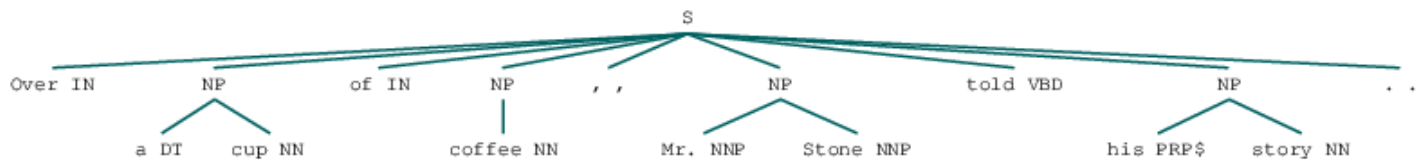
Sentences from Conll2000:

In [5]:

```
1 sentence = conll2000.chunked_sents('train.txt', chunk_types=['NP'])[99]
2 print(sentence)
3 sentence
```

```
(S
  Over/IN
  (NP a/DT cup/NN)
  of/IN
  (NP coffee/NN)
  ,/,
  (NP Mr./NNP Stone/NNP)
  told/VBD
  (NP his/PRP$ story/NN)
  ./.)
```

Out[5]:



Conll 2000 Corpus

Sentences from Conll2000 in IOB format:

In [6]:
```
1 from nltk import tree2conlltags
2 tree2conlltags(sentence)
```

Out[6]:
```
[('Over', 'IN', 'O'),
 ('a', 'DT', 'B-NP'),
 ('cup', 'NN', 'I-NP'),
 ('of', 'IN', 'O'),
 ('coffee', 'NN', 'B-NP'),
 (',', ',', 'O'),
 ('Mr.', 'NNP', 'B-NP'),
 ('Stone', 'NNP', 'I-NP'),
 ('told', 'VBD', 'O'),
 ('his', 'PRP$', 'B-NP'),
 ('story', 'NN', 'I-NP'),
 ('.', '.', 'O')]
```

Optional exercise

Use Machine Learning to build a chunker by learning sequences following next constraints:

- Design your own feature set to represent samples
- Use CRF as main algorithm
- Use Conll2000 corpora as training data

Optional exercise

Use Machine Learning to build a NER chunker learning sequences following next constraints:

- Design your own feature set to represent samples
- Use CRF as main algorithm
- Use Conll2003 (English or German) or Conll2002 (Spanish or Dutch) corpora as training data