Natural Language Processing (Part 2)

NLTK (Natural Language Toolkit)

Chunking

Chunking and Chinking

The Challenge: Entity Recognition (What we talking about?)

The NLTK **ne-chunker** finds Persons, Organizations and Locations

Working with the Named Entity Chunker (ne_chunk)

In [8]:

```
1  from nltk import word_tokenize, pos_tag, ne_chunk
2
3  sentence = "William works at CNN in Dallas."
4
5  tags = pos_tag(word_tokenize(sentence))
6  print (tags)
7
8  neChunks = ne_chunk(tags)
9  print ("\n", type(neChunks))
10 print (neChunks)
```

```
[('William', 'NNP'), ('works', 'VBZ'), ('at', 'IN'), ('CNN', 'NNP'), ('in',
'IN'), ('Dallas', 'NNP'), ('.', '.')]

 <class 'nltk.tree.Tree'>
(S
  (PERSON William/NNP)
  works/VBZ
  at/IN
  (ORGANIZATION CNN/NNP)
  in/IN
  (GPE Dallas/NNP)
  ./.)
```

ne_chunk returns a Tree with sub-trees and leaf nodes

```
In [2]:
1  sentence = "Rollo Smith knows Marla"
2
3  print (ne_chunk(pos_tag(word_tokenize(sentence))) )
```

```
(S
  (PERSON Rollo/NNP)
  (PERSON Smith/NNP)
  knows/VBZ
  (PERSON Marla/NNP))
```

Note that each part of a name is treated as a separate Person.

```
7  print ("\n Search for a subtree with label =  PERSON")
8  for i in neChunk.subtrees(filter=lambda x: x.label() == 'PERSON'):
9      print ("Subtree = ", i)
10     for node in i:
11         name, type = node
12         print ("We found a Person:", name)
13
14
```

Navigating the Tree and finding subtrees. Use a lambda to find the kind of subtree you need

```
 neChunked Sentence -- a nltk.tree.Tree
(S
  (PERSON James/NNP)
  lives/VBZ
  in/IN
  (GPE Dallas/NNP)
  ,/,
  works/VBZ
  at/IN
  (ORGANIZATION Google/NNP)
  and/CC
  knows/VBZ
  (PERSON Marla/NNP)
  ./.)

 Search for a subtree with label =  PERSON
Subtree =  (PERSON James/NNP)
We found a Person: James
Subtree =  (PERSON Marla/NNP)
We found a Person: Marla
```

Use this information to create triples. We know the rdf:type and the name of the entity

```
# do entity recognition - Organizations
sentence = "James lives in Dallas, works at Google and knows Marla."
neChunk = ne_chunk(pos_tag(word_tokenize(sentence)))
print ("\n neChunked Sentence -- a nltk.tree.Tree")
print (neChunk)

print ("\n Search for a subtree with label =  ORGANIZATION ")
for i in neChunk.subtrees(filter=lambda x: x.label() == 'ORGANIZATION'):
    print ("Subtree = ", i)
    for node in i:
        name, type = node
        print ("We found an Organization:", name)
```

```
 neChunked Sentence -- a nltk.tree.Tree
(S
  (PERSON James/NNP)
  lives/VBZ
  in/IN
  (GPE Dallas/NNP)
  ,/,
  works/VBZ
  at/IN
  (ORGANIZATION Google/NNP)
  and/CC
  knows/VBZ
  (PERSON Marla/NNP)
  ./.)

 Search for a subtree with label =  ORGANIZATION
Subtree =  (ORGANIZATION Google/NNP)
We found an Organization: Google
```

Chunking

```
1  rawtext = "Jimmy was seen dancing with Marla. "
2
3  tagged_sentence = (pos_tag(word_tokenize(rawtext)))
4  print (tagged_sentence)
5
6  from nltk.chunk import RegexpParser
7
8  chunker = RegexpParser(r'''
9  NP: {<DT>?<NN.*>+}
10 VP: {<VB.*>*<TO|IN>?}
11 ''')
12
13 ts = chunker.parse(tagged_sentence)
14 print ("\n Regex Chunker Output-----")
15 print (type(ts))
16 print (ts)
17
```

Multiple rules must be on different lines

```
[('Jimmy', 'NNP'), ('was', 'VBD'), ('seen', 'VBN'), ('dancing', 'VBG'), ('with', 'IN'), ('Marla',
'NNP'), ('.', '.')]

 Regex Chunker Output-----
<class 'nltk.tree.Tree'>
(S
  (NP Jimmy/NNP)
  (VP was/VBD seen/VBN dancing/VBG with/IN)
  (NP Marla/NNP)
  ./.)
```

OK, we got the VP (Verb Phrase) but we don't want to see extra non-verb related terms. To make it easier to process, lets end up with only Verbs

Chink our Chunks
(remove terms from our Chunk)

```
1  rawtext = "Jimmy was seen dancing with Marla. "
2  tagged_sentence = (pos_tag(word_tokenize(rawtext)))
3  print (tagged_sentence)
4
5  from nltk.chunk import RegexpParser
6
7  chunker = RegexpParser(r'''
8  NP: {<DT>?<NN.*>+}
9  VP: {<VB.*>*<TO|IN>?}
10 }<TO|IN>{
11 ''')
12
13 ts = chunker.parse(tagged_sentence)
14 print ("\n Regex Chunker Output-----")
15 print (type(ts))
16 print (ts)
```

CHINK is defined by flipped curly braces: } ... {
SAYS: Do not include TO or IN in subtree, even though I want to use them to define a VP

```
[('Jimmy', 'NNP'), ('was', 'VBD'), ('seen', 'VBN'), ('dancing', 'VBG'),
('with', 'IN'), ('Marla', 'NNP'), ('.', '.')]

 Regex Chunker Output-----
<class 'nltk.tree.Tree'>
(S
  (NP Jimmy/NNP)
  (VP was/VBD seen/VBN dancing/VBG)
  with/IN
  (NP Marla/NNP)
  ./.)
```