NLP_2019_IE

November 18, 2019

1 NLP Lab - Information Extraction

Figure 1 shows the architecture for a simple information extraction system. It begins by processing a document using several procedures: first, the raw text of the document is split into sentences using a sentence segmenter, and each sentence is further subdivided into words using a tokenizer. Next, each sentence is tagged with part-of-speech tags, which will prove very helpful in the next step, named entity detection. In this step, we search for mentions of potentially interesting entities in each sentence. Finally, we use relation detection to search for likely relations between different entities in the text.

This lab is based on the Information Extraction chapter of the NLTK book.

Figure 1: Simple Pipeline Architecture for an Information Extraction System

1.1 Loading a corpus

Example 1

This program displays three statistics for each text: average word length, average sentence length, and the number of times each vocabulary item appears in the text on average (our lexical diversity score). Observe that average word length appears to be a general property of English, since it has a recurrent value of 4. (In fact, the average word length is really 3 not 4, since the num_chars variable counts space characters.) By contrast average sentence length and lexical diversity appear to be characteristics of particular authors.

[14]:
```
import nltk, re, pprint
from nltk.corpus import gutenberg

#nltk.download('gutenberg')
#nltk.download('punkt')

for fileid in gutenberg.fileids():
    num_chars = len(gutenberg.raw(fileid))
    num_words = len(gutenberg.words(fileid))
    num_sents = len(gutenberg.sents(fileid))
    num_vocab = len(set(w.lower() for w in gutenberg.words(fileid)))
    print(round(num_chars/num_words), round(num_words/num_sents),␣
 ↪round(num_words/num_vocab), fileid)
```

```
5 25 26 austen-emma.txt
5 26 17 austen-persuasion.txt
5 28 22 austen-sense.txt
4 34 79 bible-kjv.txt
5 19 5 blake-poems.txt
4 19 14 bryant-stories.txt
4 18 12 burgess-busterbrown.txt
4 20 13 carroll-alice.txt
5 20 12 chesterton-ball.txt
5 23 11 chesterton-brown.txt
5 18 11 chesterton-thursday.txt
4 21 25 edgeworth-parents.txt
5 26 15 melville-moby_dick.txt
5 52 11 milton-paradise.txt
4 12 9 shakespeare-caesar.txt
4 12 8 shakespeare-hamlet.txt
4 12 7 shakespeare-macbeth.txt
5 36 12 whitman-leaves.txt
```

Exercise 1

Print the 1116th sentence from Macbeth using the Gutenberg corpus.

[15]:
```python
macbeth_sentences = gutenberg.sents('shakespeare-macbeth.txt')
macbeth_sentences[1116]
```

[15]:
```
['Double',
 ',',
 'double',
 ',',
 'toile',
 'and',
 'trouble',
 ';',
 'Fire',
 'burne',
 ',',
 'and',
 'Cauldron',
 'bubble']
```

1.2 Preprocessing

Example 2

In this example we will tokenize and tag a sentence with part-of-speech tags then print the annotated sentence.

[13]:
```
#nltk.download('averaged_perceptron_tagger')

sentence = "I will meet John Smith to visit Oracle headquarters tomorrow morning.
 ↪";
tokens = nltk.word_tokenize(sentence)
pos_tags = nltk.pos_tag(tokens)
print (pos_tags)
```

```
[('I', 'PRP'), ('will', 'MD'), ('meet', 'VB'), ('John', 'NNP'), ('Smith',
'NNP'), ('to', 'TO'), ('visit', 'VB'), ('Oracle', 'NNP'), ('headquarters',
'NNS'), ('tomorrow', 'NN'), ('morning', 'NN'), ('.', '.')]
```

Next, we will load sentences from one of the categories of the Brown Corpus. To do this, first we display all the categories from the corpus.

[18]:
```
from nltk.corpus import brown

#nltk.download('brown')

brown.categories()
```

[18]:
```
['adventure',
 'belles_lettres',
 'editorial',
 'fiction',
 'government',
 'hobbies',
 'humor',
 'learned',
 'lore',
 'mystery',
 'news',
 'religion',
 'reviews',
 'romance',
 'science_fiction']
```

Exercise 2

POS tag the first three sentences of the Brown Corpus that are in the category ‘news’ and then print the original sentences and the tagged sentences.

[23]:
```
from nltk.corpus import brown

sentences=brown.sents(categories='news')

for i in range(0, 3):
    # Print the original sentence
    print ("Sentence " + str(i) + ":")
    sent = sentences[i]
    print (sent)
    print ("\n")

    # Print the tagged sentence with POS
    print ("Sentence with POS:")
    # Your code goes here
    sent = nltk.pos_tag(sent)
    print (sent)
    print ("\n")
```

```
Sentence 0:
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an',
'investigation', 'of', "Atlanta's", 'recent', 'primary', 'election', 'produced',
'``', 'no', 'evidence', "''", 'that', 'any', 'irregularities', 'took', 'place',
'.']


Sentence with POS:
[('The', 'DT'), ('Fulton', 'NNP'), ('County', 'NNP'), ('Grand', 'NNP'), ('Jury',
'NNP'), ('said', 'VBD'), ('Friday', 'NNP'), ('an', 'DT'), ('investigation',
'NN'), ('of', 'IN'), ("Atlanta's", 'NNP'), ('recent', 'JJ'), ('primary', 'JJ'),
('election', 'NN'), ('produced', 'VBD'), ('``', '``'), ('no', 'DT'),
('evidence', 'NN'), ("''", "''"), ('that', 'IN'), ('any', 'DT'),
('irregularities', 'NNS'), ('took', 'VBD'), ('place', 'NN'), ('.', '.')]


Sentence 1:
['The', 'jury', 'further', 'said', 'in', 'term-end', 'presentments', 'that',
'the', 'City', 'Executive', 'Committee', ',', 'which', 'had', 'over-all',
'charge', 'of', 'the', 'election', ',', '``', 'deserves', 'the', 'praise',
'and', 'thanks', 'of', 'the', 'City', 'of', 'Atlanta', "''", 'for', 'the',
'manner', 'in', 'which', 'the', 'election', 'was', 'conducted', '.']


Sentence with POS:
[('The', 'DT'), ('jury', 'NN'), ('further', 'RB'), ('said', 'VBD'), ('in',
'IN'), ('term-end', 'JJ'), ('presentments', 'NNS'), ('that', 'IN'), ('the',
'DT'), ('City', 'NNP'), ('Executive', 'NNP'), ('Committee', 'NNP'), (',', ','),
('which', 'WDT'), ('had', 'VBD'), ('over-all', 'JJ'), ('charge', 'NN'), ('of',
'IN'), ('the', 'DT'), ('election', 'NN'), (',', ','), ('``', '``'), ('deserves',
'VBZ'), ('the', 'DT'), ('praise', 'NN'), ('and', 'CC'), ('thanks', 'NNS'),
('of', 'IN'), ('the', 'DT'), ('City', 'NNP'), ('of', 'IN'), ('Atlanta', 'NNP'),
("''", "''"), ('for', 'IN'), ('the', 'DT'), ('manner', 'NN'), ('in', 'IN'),
('which', 'WDT'), ('the', 'DT'), ('election', 'NN'), ('was', 'VBD'),
('conducted', 'VBN'), ('.', '.')]
```

```
Sentence 2:
['The', 'September-October', 'term', 'jury', 'had', 'been', 'charged', 'by',
'Fulton', 'Superior', 'Court', 'Judge', 'Durwood', 'Pye', 'to', 'investigate',
'reports', 'of', 'possible', '``', 'irregularities', "''", 'in', 'the', 'hard-
fought', 'primary', 'which', 'was', 'won', 'by', 'Mayor-nominate', 'Ivan',
'Allen', 'Jr.', '.']


Sentence with POS:
[('The', 'DT'), ('September-October', 'NNP'), ('term', 'NN'), ('jury', 'NN'),
('had', 'VBD'), ('been', 'VBN'), ('charged', 'VBN'), ('by', 'IN'), ('Fulton',
'NNP'), ('Superior', 'NNP'), ('Court', 'NNP'), ('Judge', 'NNP'), ('Durwood',
'NNP'), ('Pye', 'NNP'), ('to', 'TO'), ('investigate', 'VB'), ('reports', 'NNS'),
('of', 'IN'), ('possible', 'JJ'), ('``', '``'), ('irregularities', 'NNS'),
("''", "''"), ('in', 'IN'), ('the', 'DT'), ('hard-fought', 'JJ'), ('primary',
'NN'), ('which', 'WDT'), ('was', 'VBD'), ('won', 'VBN'), ('by', 'IN'), ('Mayor-
nominate', 'NNP'), ('Ivan', 'NNP'), ('Allen', 'NNP'), ('Jr.', 'NNP'), ('.',
'.')]
```

1.3 Chunking

Chunking is a basic technique used for entity detection is chunking, which segments and labels multi-token sequences as illustrated in Figure 2.

Figure 2: Segmentation and Labeling at both the Token and Chunk Levels

To find the chunk structure for a given sentence, the RegexpParser chunker begins with a flat structure in which no tokens are chunked. The chunking rules are applied in turn, successively updating the chunk structure. Once all of the rules have been invoked, the resulting chunk structure is returned.

Example 3: Noun Phrase Chunking

In order to create an NP-chunker, we will first define a chunk grammar, consisting of rules that indicate how sentences should be chunked. In this case, we will define a simple grammar with a single regular-expression rule. This rule says that an NP chunk should be formed whenever the chunker finds an optional determiner (DT) followed by any number of adjectives (JJ) and then a noun (NN). Using this grammar, we create a chunk parser, and test it on our example sentence. The result is a tree, which we can either print, or display graphically.

[27]:
```python
from nltk.corpus import brown

grammar = "NP: {<DT>?<JJ>*<NN>}" # simple grammar
cp = nltk.RegexpParser(grammar) # create a chunk parser using this grammar

sentences=brown.sents(categories='news')
```

```
for i in range(0, 3):
    sent = nltk.pos_tag(sentences[i])
    result = cp.parse(sent) # the result is a tree

    print (result) # print the tree

    #result.draw() # draw graphically
```

```
(S
  The/DT
  Fulton/NNP
  County/NNP
  Grand/NNP
  Jury/NNP
  said/VBD
  Friday/NNP
  (NP an/DT investigation/NN)
  of/IN
  Atlanta's/NNP
  (NP recent/JJ primary/JJ election/NN)
  produced/VBD
  ``/``
  (NP no/DT evidence/NN)
  ''/''
  that/IN
  any/DT
  irregularities/NNS
  took/VBD
  (NP place/NN)
  ./.)
(S
  (NP The/DT jury/NN)
  further/RB
  said/VBD
  in/IN
  term-end/JJ
  presentments/NNS
  that/IN
  the/DT
  City/NNP
  Executive/NNP
  Committee/NNP
  ,/,
  which/WDT
  had/VBD
  (NP over-all/JJ charge/NN)
  of/IN
  (NP the/DT election/NN)
  ,/,
  ``/``
  deserves/VBZ
  (NP the/DT praise/NN)
  and/CC
  thanks/NNS
  of/IN
  the/DT
  City/NNP
  of/IN
  Atlanta/NNP
  ''/''
  for/IN
  (NP the/DT manner/NN)
  in/IN
  which/WDT
  (NP the/DT election/NN)
  was/VBD
  conducted/VBN
  ./.)
(S
  The/DT
  September-October/NNP
  (NP term/NN)
  (NP jury/NN)
  had/VBD
  been/VBN
  charged/VBN
  by/IN
  Fulton/NNP
  Superior/NNP
  Court/NNP
  Judge/NNP
  Durwood/NNP
  Pye/NNP
  to/TO
  investigate/VB
  reports/NNS
  of/IN
  possible/JJ
  ``/``
  irregularities/NNS
  ''/''
  in/IN
  (NP the/DT hard-fought/JJ primary/NN)
  which/WDT
```

```
was/VBD
won/VBN
by/IN
Mayor-nominate/NNP
Ivan/NNP
Allen/NNP
Jr./NNP
./.)
```

Exercise 3: Verb Phrase Chunking

Create a verb phrase chunker for the following pattern “verb to verb” that covers verb phrases such as “serve to protect” and “like to see”. Run this chunker on the first 100 sentences of the Brown Corpus.

[39]:
```python
import nltk

# Your code goes here
grammar = "CHUNK: {<V.*> <TO> <V.*>}"

cp = nltk.RegexpParser(grammar)

brown = nltk.corpus.brown
sentences = brown.tagged_sents()

for i in range(0, 100):
     tree = cp.parse(sentences[i])
     for subtree in tree.subtrees():
         if subtree.label() == 'CHUNK': print(subtree)
```

```
(CHUNK combined/VBN to/TO achieve/VB)
(CHUNK continue/VB to/TO place/VB)
(CHUNK serve/VB to/TO protect/VB)
(CHUNK wanted/VBD to/TO wait/VB)
(CHUNK allowed/VBN to/TO place/VB)
(CHUNK expected/VBN to/TO become/VB)
(CHUNK expected/VBN to/TO approve/VB)
(CHUNK expected/VBN to/TO make/VB)
(CHUNK intends/VBZ to/TO make/VB)
(CHUNK seek/VB to/TO set/VB)
(CHUNK like/VB to/TO see/VB)
```

1.4 Named Entity Recognition

NLTK provides a classifier that has already been trained to recognize named entities, accessed with the function nltk.ne_chunk(). If we set the parameter binary=True [1], then named entities are just tagged as NE; otherwise, the classifier adds category labels such as PERSON, ORGANIZATION, and GPE.

Example 4

Annotate a sentence with named entities and print the result.

```
[44]: nltk.download('maxent_ne_chunker')
      nltk.download('words')

      sentence = "I will meet John Smith to visit Oracle headquarters tomorrow␣
       ↪morning";
      tokens = nltk.word_tokenize(sentence)
      pos_tags = nltk.pos_tag(tokens)

      print (nltk.ne_chunk(pos_tags))
```

```
(S
  I/PRP
  will/MD
  meet/VB
  (PERSON John/NNP Smith/NNP)
  to/TO
  visit/VB
  (GPE Oracle/NNP)
  headquarters/NNS
  tomorrow/NN
  morning/NN)
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /home/gb5/nltk_data…
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to /home/gb5/nltk_data…
[nltk_data]   Package words is already up-to-date!
```

Exercise 4

Extract all the unique named entities from the first 20 sentences of the of the Brown Corpus that are in the category ‘news’.

```
[49]: from nltk.corpus import brown

      def extract_entity_names(t):
          # Your code goes here
          entity_names = [' '.join(map(lambda x: x[0], ne.leaves())) for ne in t if␣
       ↪isinstance(ne, nltk.tree.Tree)]

          return entity_names

      news_sentences=brown.sents(categories='news')

      entity_names = []
```

```
for i in range(0, 20):
  tagged_sent = nltk.pos_tag(news_sentences[i])
  chunked_sent = nltk.ne_chunk(tagged_sent, binary=True)

  # Print results per sentence
  print (extract_entity_names(chunked_sent))
  entity_names.extend(extract_entity_names(chunked_sent))

# Print all entity names
print (entity_names)

# Print unique entity names
print (set(entity_names))
```

```
['Fulton County Grand Jury']
['City Executive Committee', 'Atlanta']
['Fulton Superior Court']
[]
[]
['Fulton']
['Atlanta', 'Fulton County']
['Merger']
[]
['City Purchasing Department']
[]
[]
[]
[]
['Fulton County', 'Fulton County']
[]
['Fulton County']
['Fulton']
['Fulton']
[]
['Fulton County Grand Jury', 'City Executive Committee', 'Atlanta', 'Fulton
Superior Court', 'Fulton', 'Atlanta', 'Fulton County', 'Merger', 'City
Purchasing Department', 'Fulton County', 'Fulton County', 'Fulton County',
'Fulton', 'Fulton']
{'Merger', 'City Purchasing Department', 'City Executive Committee', 'Atlanta',
'Fulton County Grand Jury', 'Fulton County', 'Fulton Superior Court', 'Fulton'}
```

1.5 Relation Extraction

Once named entities have been identified in a text, we then want to extract the relations that exist between them. We will typically be looking for relations between specified types of named entity. One way of approaching this task is to initially look for all triples of the form (X, , Y), where X and Y are named entities of the required types, and is the string of words that intervenes between

X and Y. We can then use regular expressions to pull out just those instances of that express the relation that we are looking for.

Example 5

The following example searches for strings that contain the word in. The special regular expression (?!.+ing) is a negative lookahead assertion that allows us to disregard strings such as success in supervising the transition of, where in is followed by a gerund. For this example we will use the nltk.corpus.ieer.

[51]:
```
import re, nltk

nltk.download('ieer')

IN = re.compile(r'.*\bin\b(?!\b.+ing\b)')

for doc in nltk.corpus.ieer.parsed_docs('NYT_19980315'):
    for rel in nltk.sem.relextract.extract_rels('ORG', 'LOC',␣
↪doc,corpus='ieer', pattern = IN):
        print (nltk.sem.relextract.rtuple(rel))
```

```
[nltk_data] Downloading package ieer to /home/gb5/nltk_data…
```

```
[ORG: 'WHYY'] 'in' [LOC: 'Philadelphia']
[ORG: 'McGlashan &AMP; Sarrail'] 'firm in' [LOC: 'San Mateo']
[ORG: 'Freedom Forum'] 'in' [LOC: 'Arlington']
[ORG: 'Brookings Institution'] ', the research group in' [LOC: 'Washington']
[ORG: 'Idealab'] ', a self-described business incubator based in' [LOC: 'Los
Angeles']
[ORG: 'Open Text'] ', based in' [LOC: 'Waterloo']
[ORG: 'WGBH'] 'in' [LOC: 'Boston']
[ORG: 'Bastille Opera'] 'in' [LOC: 'Paris']
[ORG: 'Omnicom'] 'in' [LOC: 'New York']
[ORG: 'DDB Needham'] 'in' [LOC: 'New York']
[ORG: 'Kaplan Thaler Group'] 'in' [LOC: 'New York']
[ORG: 'BBDO South'] 'in' [LOC: 'Atlanta']
[ORG: 'Georgia-Pacific'] 'in' [LOC: 'Atlanta']
```

```
[nltk_data]   Unzipping corpora/ieer.zip.
```

Exercise 5

Extract places of birth of people from the the ieeer corpus, using the ‘X born in Y’ pattern, where X is a person and Y is a location.

[53]:
```
from nltk.corpus import ieer

# Your code goes here
BORNIN = re.compile(r'.*\bborn in\b')

for fileId in ieer.fileids():
    for doc in nltk.corpus.ieer.parsed_docs(fileId):
         for rel in nltk.sem.relextract.extract_rels('PER', 'LOC', doc,␣
↪corpus='ieer', pattern = BORNIN):
                print(nltk.sem.relextract.rtuple(rel))
```

```
[PER: 'McCarthy'] 'was born in' [LOC: 'Belle Plaine']
```

Example 6

In this example we extract people and their role in an organisation using a predefined list of roles.

[54]:
```
def roles_demo(trace=0):
    from nltk.corpus import ieer
    roles = """
    (.*(                       # assorted roles
    analyst|
    chair(wo)?man|
    commissioner|
    counsel|
    director|
    economist|
    editor|
    executive|
    foreman|
    governor|
    head|
    lawyer|
    leader|
    librarian).*)|
    manager|
    partner|
    president|
    producer|
    professor|
    researcher|
    spokes(wo)?man|
    writer|
    ,\sof\sthe?\s*  # "X, of (the) Y"
    """
    ROLES = re.compile(roles, re.VERBOSE)

    print()
    print("IEER: has_role(PER, ORG) -- raw rtuples:")
    print("=" * 45)

    for file in ieer.fileids():
        for doc in ieer.parsed_docs(file):
            lcon = rcon = False
            if trace:
                print(doc.docno)
                print("=" * 15)
                lcon = rcon = True
            for rel in nltk.sem.relextract.extract_rels('PER', 'ORG', doc,␣
↪corpus='ieer', pattern=ROLES):
                print(nltk.sem.relextract.rtuple(rel, lcon=lcon, rcon=rcon))

roles_demo()
```

```
IEER: has_role(PER, ORG) -- raw rtuples:
=============================================
[PER: 'Kivutha Kibwana'] ', of the' [ORG: 'National Convention Assembly']
[PER: 'Boban Boskovic'] ', chief executive of the' [ORG: 'Plastika']
[PER: 'Annan'] ', the first sub-Saharan African to head the' [ORG: 'United
Nations']
[PER: 'Kiriyenko'] 'became a foreman at the' [ORG: 'Krasnoye Sormovo']
[PER: 'Annan'] ', the first sub-Saharan African to head the' [ORG: 'United
Nations']
[PER: 'Mike Godwin'] ', chief counsel for the' [ORG: 'Electronic Frontier
Foundation']
[PER: 'Robert Mergess'] ', the co-director of the' [ORG: 'Berkeley Center for
Law and Technology']
[PER: 'Jack Balkin'] ", director of the school's program. ``What happened at"
[ORG: 'Yale']
[PER: 'William Gale'] ', an economist at the' [ORG: 'Brookings Institution']
[PER: 'Joel Slemrod'] ', an economist at the' [ORG: 'University of Michigan']
[PER: 'Alan Braverman'] ', Internet analyst at' [ORG: 'Credit Suisse First
Boston']
[PER: 'Michael Coffey'] ', managing editor of' [ORG: 'Publishers Weekly']
[PER: 'Lorne Michaels'] ", the executive producer of ``Saturday Night Live.''"
[ORG: 'TV Books']
[PER: 'James Billington'] ', the librarian of' [ORG: 'Congress']
[PER: 'Sherry Lansing'] ', chairwoman of the' [ORG: 'Paramount Motion Picture
Group']
[PER: 'Charlotte Forest'] ', executive producer for' [ORG: 'Homestead
Editorial']
[PER: 'John Wren'] ', the president and chief executive at' [ORG: 'Omnicom']
[PER: 'Charlie Moss'] ', the chairman of' [ORG: 'Moss/Dragoti']
[PER: 'Norio Ohga'] ', chairman of the' [ORG: 'Sony Corporation']
[PER: 'Wendy Beale Needham'] ', an auto analyst at' [ORG: 'Donaldson, Lufkin
&AMP; Jenrette']
[PER: 'David Bradley'] ', an auto analyst at' [ORG: 'J.P. Morgan Securities
Inc.']
[PER: 'Michael W. McCarthy'] ', a former chairman of' [ORG: 'Merrill Lynch &AMP;
Co.']
```

```
[PER: 'McCarthy'] 'was a governor of the' [ORG: 'American Stock Exchange']
[PER: 'Anthony Chan'] ', managing director and chief economist at the' [ORG:
'Banc One Investment Advisors Corp.']
[PER: 'Paul Volcker'] ', former chairman of the' [ORG: 'U.S. Federal Reserve']
[PER: 'Katherine Abraham'] ', commissioner of the' [ORG: 'Bureau of Labor
Statistics']
[PER: 'David Wyss'] ', chief economist at' [ORG: "Standard &AMP; Poor's DRI"]
[PER: 'Bruce Steinberg'] ', chief economist at' [ORG: 'Merrill Lynch']
[PER: 'Mitchell Fromstein'] ', head of' [ORG: 'Manpower Inc.']
[PER: 'Barry Travis'] ', executive director of the' [ORG: 'Little Rock
Convention and Visitors Bureau']
[PER: 'Pamela Bethel'] ', a lawyer for' [ORG: 'Kramerbooks']
[PER: 'Walter'] ', who now heads the' [ORG: 'Waterford Foundation']
[PER: 'Alan Horn'] ', the chairman of' [ORG: 'Castle Rock Entertainment']
[PER: 'Dan Morgenstern'] ', director of the' [ORG: 'Institute of Jazz Studies']
[PER: 'Robert Schelin'] ', acting managing director of' [ORG: 'Smithsonian
Press/Smithsonian Productions']
[PER: 'Peter Cannell'] ', director of the' [ORG: 'Smithsonian Institution
Press']
[PER: 'Trimble'] ', the leader of the Protestant' [ORG: 'Ulster Unionist Party']
[PER: 'Seamus Mallon'] ', deputy leader of the' [ORG: 'Social Democrats']
[PER: 'Alexander Shokhin'] ', the parliamentary leader of' [ORG: 'Our Home Is
Russia']
[PER: 'Yegor Stroyev'] ', the head of the' [ORG: 'upper house']
```

Exercise 6

Extract people and their role in an organisation by using the ‘X ROLE at the Y’ or ‘X ROLE of the Y’ patterns, where X is a person and Y is an organisation.

[56]:
```python
from nltk.corpus import ieer

# Your code goes here
ROLES = re.compile(',.*(\sat\sthe?|\sof\sthe?)')

for file in ieer.fileids():
    for doc in nltk.corpus.ieer.parsed_docs(file):
        for r in nltk.sem.relextract.extract_rels('PER', 'ORG', doc,␣
 ↪corpus='ieer', pattern=ROLES):
            #print  (nltk.sem.relextract.clause(r, relsym="ROLES"))
            print (nltk.sem.relextract.rtuple(r))
```

```
[PER: 'Kivutha Kibwana'] ', of the' [ORG: 'National Convention Assembly']
[PER: 'Boban Boskovic'] ', chief executive of the' [ORG: 'Plastika']
[PER: 'Robert Mergess'] ', the co-director of the' [ORG: 'Berkeley Center for
Law and Technology']
[PER: 'Jack Balkin'] ", director of the school's program. ``What happened at"
[ORG: 'Yale']
[PER: 'David Post'] ', co-founder of the' [ORG: 'Cyberspace Law Institute']
```

```
[PER: 'William Gale'] ', an economist at the' [ORG: 'Brookings Institution']
[PER: 'Joel Slemrod'] ', an economist at the' [ORG: 'University of Michigan']
[PER: 'Kaufman'] ', president of the privately held' [ORG: 'TV Books LLC']
[PER: 'Sherry Lansing'] ', chairwoman of the' [ORG: 'Paramount Motion Picture
Group']
[PER: 'Rick Yorn'] ', his manager at the firm' [ORG: 'Addis-Wechsler &AMP;
Associates']
[PER: 'Ken Kaess'] ', president of the' [ORG: 'DDB Needham']
[PER: 'Norio Ohga'] ', chairman of the' [ORG: 'Sony Corporation']
[PER: 'Raymond Rosen'] ', a sex researcher and professor of psychiatry at the'
[ORG: 'Robert Wood Johnson Medical School']
[PER: 'Pepper Schwartz'] ', a sociology professor at the' [ORG: 'University of
Washington']
[PER: 'Irwin Goldstein'] ', a professor of urology at the' [ORG: 'Boston
University School of Medicine']
[PER: 'Jennifer Berman'] ', a urologist at the' [ORG: 'University of Maryland']
[PER: 'Anthony Chan'] ', managing director and chief economist at the' [ORG:
'Banc One Investment Advisors Corp.']
[PER: 'Kevin Ashby'] ', publisher of the local newspaper,' [ORG: 'The Sun
Advocate']
[PER: 'Paul Volcker'] ', former chairman of the' [ORG: 'U.S. Federal Reserve']
[PER: 'Israel Singer'] ', the secretary-general of the' [ORG: 'World Jewish
Congress']
[PER: 'Katherine Abraham'] ', commissioner of the' [ORG: 'Bureau of Labor
Statistics']
[PER: 'Lloyd Kiva New'] ', president emeritus of the' [ORG: 'Institute of
American Indian Art']
[PER: 'Barry Travis'] ', executive director of the' [ORG: 'Little Rock
Convention and Visitors Bureau']
[PER: 'Linda Ward'] ', general manager of the' [ORG: 'Legacy Hotel']
[PER: 'J. Jackson Walter'] ', who served as president of the' [ORG: 'National
Trust for Historic Preservation']
[PER: 'Dan Morgenstern'] ', director of the' [ORG: 'Institute of Jazz Studies']
[PER: 'Peter Cannell'] ', director of the' [ORG: 'Smithsonian Institution
Press']
[PER: 'Trimble'] ', the leader of the Protestant' [ORG: 'Ulster Unionist Party']
[PER: 'Seamus Mallon'] ', deputy leader of the' [ORG: 'Social Democrats']
[PER: 'Yegor Stroyev'] ', the head of the' [ORG: 'upper house']
```

1.6 The WordNet hierarchy

WordNet is a semantically-oriented dictionary of English, similar to a traditional thesaurus but with a richer structure. NLTK includes the English WordNet, with 155,287 words and 117,659 synonym sets. Concepts are hierarchically organised using hypernym-hyponym relations.

Example 7

Given a concept like feline, we can look at the concepts that are more specific; the (immediate) hyponyms. We can also navigate up the hierarchy by visiting hypernyms. Some words have multiple paths, because they can be classified in more than one way. There are two paths between car.n.01 and entity.n.01 because wheeled_vehicle.n.01 can be classified as both a vehicle and a container. Some words have multiple paths, because they can be classified in more than one way. There are two paths between car.n.01 and entity.n.01 because wheeled_vehicle.n.01 can be classified as both a vehicle and a container.

[58]:
```
from nltk.corpus import wordnet as wn

nltk.download('wordnet')

carnivore = wn.synset('carnivore.n.01')
carnivore_isa = carnivore.hypernyms()
carnivore_isa[0]
sorted(lemma.name() for synset in carnivore_isa for lemma in synset.lemmas())
```

```
[nltk_data] Downloading package wordnet to /home/gb5/nltk_data…
[nltk_data]   Unzipping corpora/wordnet.zip.
```

[58]:
```
['eutherian', 'eutherian_mammal', 'placental', 'placental_mammal']
```

Exercise 7

List the concepts that are more specific than the noun carnivore, in other words its (immediate) hyponyms.

[59]:
```
from nltk.corpus import wordnet as wn
carnivore = wn.synset('carnivore.n.01')

# Your code goes here
types_of_carnivore = carnivore.hyponyms()
types_of_carnivore[0]
sorted(lemma.name() for synset in types_of_carnivore for lemma in synset.
 ↪lemmas())
```

[59]:
```
['bear',
 'canid',
 'canine',
 'felid',
 'feline',
 'fissiped',
 'fissiped_mammal',
 'mustelid',
 'musteline',
 'musteline_mammal',
 'procyonid',
 'viverrine',
 'viverrine_mammal']
```

Example 8

Some words have multiple paths, because they can be classified in more than one way. There are two paths between car.n.01 and entity.n.01 because wheeled_vehicle.n.01 can be classified as both a vehicle and a container.

[61]:
```
car=wn.synset('car.n.01')
types_of_car=car.hypernyms()
paths = car.hypernym_paths()
len(paths) # display the number of paths to root

#[synset.name() for synset in paths[0]] #display the first path

[synset.name() for synset in paths[1]] #display the second path
```

[61]:
```
['entity.n.01',
 'physical_entity.n.01',
 'object.n.01',
 'whole.n.02',
 'artifact.n.01',
 'instrumentality.n.03',
 'conveyance.n.03',
 'vehicle.n.01',
 'wheeled_vehicle.n.01',
 'self-propelled_vehicle.n.01',
 'motor_vehicle.n.01',
 'car.n.01']
```

[ ]:
```

```