Hypernym Relationship Extraction

In this example, we will use NLTK and Hearst Pattern for hypernym relationship extraction.

- Firstly, install python environment
- Install NLTK: pip install nltk
- Download data distribution for NLTK. Install using NLTK downloader: `nltk.download()` . If cannot download using `nltk.download()` , try download manually from https://github.com/nltk/nltk_data/tree/gh-pages![image.png](attachment:image.png) (https://github.com/nltk/nltk_data/tree/gh-pages!%5Bimage.png%5D(attachment:image.png)) or https://pan.baidu.com/s/1wONWpaa86_wnsIksKda8eQ (https://pan.baidu.com/s/1wONWpaa86_wnsIksKda8eQ) (code:tfon)
- Unzip the downloaded file to the following folder: `nltk.data.find(".")`
- Unzip each zip file in the ten folders: *chunkers, corpora, grammers, help, misc, models, sentiment, stemmers, taggers, tokenizers*

Hyponym Extraction using Hearst Pattern

Hyponym extraction follows the following 4 steps:

- Noun phrase chunking or named eneity chunking. You can use any np chunking/named entity technique.
- Chunked sentences prepare. Traverse the chunked result, if the label is `NP` , then merge all the words in this chunk and add a prefix `NP_` (for subsequence process).
- Chunking refinement. If two or more NPs next to each other should be merged into a single NP. Eg., *"NP_foo NP_bar blah blah"* becomes *"NP_foo_bar blah blah"*
- Find the hypernym and hyponym pairs based on the refined prepared chunked sentence.

In [1]:
```
import nltk
import re
from nltk import pos_tag, word_tokenize, Tree
from nltk.stem import WordNetLemmatizer
```

Regular expression practice: In this example, we show one regex pattern example for Hearst pattern: `NP such as {NP,}* {(or | and)} NP` (https://docs.python.org/3/library/re.html (https://docs.python.org/3/library/re.html))

In [2]:

```
regex = r"(NP_\w+ (, )?such as (NP_\w+ ?(, )?(and |or )?)+)"
test_str = "NP_1 such as NP_2 , NP_3 and NP_4 "
matches = re.search(regex, test_str)
if matches:
    # Match.group([group1, ...]) Returns one or more subgroups of the match.
    # If there is a single argument, the result is a single string;
    # if there are multiple arguments, the result is a tuple with one item per argument.
    # Without arguments, group1 defaults to zero (the whole match is returned).
    print(matches.group(0))
```

```
NP_1 such as NP_2 , NP_3 and NP_4
```

Step1: Chunking Sentence

- Note the result is not the chunked np, instead is the chunk tree structure

In [3]:

```python
def np_chunking(sentence):
    # your implementation

    return result

print(np_chunking("""I like to listen to music from musical genres,such as blues,rock and jazz."""))
```

```
(S
  I/PRP
  like/VBP
  to/TO
  listen/VB
  to/TO
  (NP music/NN)
  from/IN
  (NP musical/JJ genres/NNS)
  ,/,
  such/JJ
  as/IN
  (NP blues/NNS)
  ,/,
  (NP rock/NN)
  and/CC
  (NP jazz/NN)
  ./.)
```

Step2: Prepare the chunked result for subsequent Hearst pattern matching

- Traverse the chunked result, if the label is `NP` , then merge all the words in this chunk and add a prefix `NP_`
- All the tokens are separated with a white space (`" "`)
- Remember to lemmatize words, using `WordNetLemmatizer` (`from nltk.stem import WordNetLemmatizer`)

In [4]:
```
# prepare the chunked sentence by merging words and add prefix NP_
def prepare_chunks(chunks):
        # If chunk is NP, start with NP_ and join tokens in chunk with _ ; Else just keep the token as it is
        terms = []
        for chunk in chunks:
            label = None
            try:
                # see if the chunk is simply a word or a NP. But non-NP fail on this method call
                label = chunk.label()
            except:
                pass
            # Based on the label, do processing, your implementation here...
```

In [5]:
```
raw_text = "I like to listen to music from musical genres,such as blues,rock and jazz."
chunk_res = np_chunking(raw_text)
print(prepare_chunks(chunk_res))
```

```
I like to listen to NP_music from NP_musical_genre , such as NP_blue , NP_rock and NP_jazz .
```

Step3: Refinement chunking

If two or more NPs next to each other should be merged into a single NP. E.g., `NP_foo NP_bar blah blah` becomes `NP_foo_bar blah blah`

In [6]:
```
def merge_NP(prepared_chunks):
    sentence = re.sub(r"(NP_\w+ NP_\w+)+",lambda m: m.expand(r'\1').replace(" NP_", "_"),prepared_chunks)
    return sentence
```

In [7]:
```
merge_NP("NP_foo NP_bar blah blah")
```

Out[7]:
```
'NP_foo_bar blah blah'
```

Step4: Find the hypernym and hyponyms on processed chunked results

- Define Hearst patterns. Besides the regex, we also need to specify whether the hypernym is in the first part or the second part in the pattern.
  - For example, in the pattern `NP1 such as NP2 AND NP3` , the hypernym is the first part of the pattern; in the pattern `NP1 , NP2 and other NP3` , the hypernym is the last part of the pattern.

- After regex matching, find all the NPs and extract the hypernym and hyponym pairs based on the `first` or `last` attribute.
- Clean the NPs by removing the prefix `NP_` and `_`

In [8]:
```
# Given by the prepared text, return the hypernym-hyponym pairs
def hyponym_extract(prepared_text, hearst_patterns):
    # your implementation
    return pairs

hearst_patterns = [("(NP_\w+ (, )?such as (NP_\w+ ?(, )?(and |or )?)+)", "first"),
                   ("((NP_\w+ ?(, )?)+(and |or )?other NP_\w+)","last")]  # two examples for hearst pattern
print(hyponym_extract(prepare_chunks(np_chunking("I like to listen to music from musical genres,such as blues,rock and
print(hyponym_extract(prepare_chunks(np_chunking("He likes to play basketball,football and other sports.")),hearst_pa
```

```
[('NP_blue', 'NP_musical_genre'), ('NP_rock', 'NP_musical_genre'), ('NP_jazz', 'NP_musical_genre')]
[('NP_basketball', 'NP__sport'), ('NP_football', 'NP__sport')]
```

In [10]:
```
def find_hyponyms(sentence, hearst_patterns):
    # your implementation

print(find_hyponyms("""I like to listen to music from musical genres,such as blues,rock and jazz.""", hearst_patterns
print(find_hyponyms("""He likes to play basketball,football and other sports.""",hearst_patterns))
```

```
[[('NP_blue', 'NP_musical_genre'), ('NP_rock', 'NP_musical_genre'), ('NP_jazz', 'NP_musical_genre')]]
[[('NP_basketball', 'NP__sport'), ('NP_football', 'NP__sport')]]
```

In [11]:
```
def clean_np(term):
    return term.replace("NP_", "").replace("_", " ")
clean_np('NP_football')
```

Out[11]: `'football'`

Complete Program for Hypernym extraction using Hearst Pattern

In [12]:

```
# Merge everything to get the final extractor
class HearstPatterns(object):
    # finish the extractor class using the aforementioned functions

```

In [13]:

```
# Test case for hearst patterns
hp = HearstPatterns()
test = ["Agar is a substance prepared from a mixture of red algae, such as Gelidium,for laboratory or industrial use.'
                          "... works by such authors as Herrick, Goldsmith, and Shakespeare.",
                          "... bistros, coffee shops, and other cheap eating places.",
                          "...all common law countries, including Canada and England.",
                          "...most European countries, especially France, England, and Spain."]
        #         text = 'I like to listen to music from musical genres such as blues, rock and jazz. He likes to play
for txt in test:
    hps = hp.find_hyponyms(txt)
    print(hps)
```

```
[('Gelidium', 'red algae')]
[('Herrick', ' author'), ('Goldsmith', ' author'), ('Shakespeare', ' author')]
[('bistro', ' cheap eating place'), ('coffee shop', ' cheap eating place')]
[('Canada', 'common law country'), ('England', 'common law country')]
[('France', 'European country'), ('England', 'European country'), ('Spain', 'European country')]
```