NLTK tutorial

(From https://www.nltk.org/ (https://www.nltk.org/)) NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

We'll talk about the following sections in this tutorial:

1. Tokenizer
2. Stemmer
3. WordNet
4. Tips to the assignments

In [1]:
```
!pip install nltk
!pip install numpy
```

```
Requirement already satisfied: nltk in /Users/tianqing/anaconda3/e
nvs/COMP4901/lib/python3.8/site-packages (3.5)
Requirement already satisfied: regex in /Users/tianqing/anaconda3/
envs/COMP4901/lib/python3.8/site-packages (from nltk) (2020.7.14)
Requirement already satisfied: click in /Users/tianqing/anaconda3/
envs/COMP4901/lib/python3.8/site-packages (from nltk) (7.1.2)
Requirement already satisfied: tqdm in /Users/tianqing/anaconda3/e
nvs/COMP4901/lib/python3.8/site-packages (from nltk) (4.49.0)
Requirement already satisfied: joblib in /Users/tianqing/anaconda3
/envs/COMP4901/lib/python3.8/site-packages (from nltk) (0.16.0)
Requirement already satisfied: numpy in /Users/tianqing/anaconda3/
envs/COMP4901/lib/python3.8/site-packages (1.19.2)
```

1. NLTK Tokenizer

In [2]:
```
import nltk
nltk.download('punkt') # to make nltk.tokenizer works
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to /Users/tianqing/nltk_dat
a...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/tianqing/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[2]:
```
True
```

In [3]:
```
text1 = "Text mining is to identify useful information."
text2 = "Current NLP models isn't able to solve NLU perfectly."

print("string.split tokenizer", text1.split(" "))
print("string.split tokenizer", text2.split(" "))
```

```
string.split tokenizer ['Text', 'mining', 'is', 'to', 'identify',
'useful', 'information.']
string.split tokenizer ['Current', 'NLP', 'models', "isn't", 'able
', 'to', 'solve', 'NLU', 'perfectly.']
```

Cannot deal with punctuations, i.e., full stops and apostrophes.

In [4]:
```
import regex # regular expression
print("regular expression tokenizer", regex.split("[\s\.]", text1))
print("regular expression tokenizer", regex.split("[\s\.]", text2))
```

```
regular expression tokenizer ['Text', 'mining', 'is', 'to', 'ident
ify', 'useful', 'information', '']
regular expression tokenizer ['Current', 'NLP', 'models', "isn't",
'able', 'to', 'solve', 'NLU', 'perfectly', '']
```

- Here, the `string.split` function can not deal with punctuations
- Simple regular expression can deal with most punctuations but may fail in the cases of "isn't, wasn't, can't"

In [5]:
```
def tokenize(text):
    """
    :param text: a doc with multiple sentences, type: str
    return a word list, type: list
    e.g.
    Input: 'Text mining is to identify useful information.'
    Output: ['Text', 'mining', 'is', 'to', 'identify', 'useful', 'in
formation', '.']
    """
    return nltk.word_tokenize(text)
```

In [6]:
```
print(tokenize(text1))
print(tokenize(text2))
```

```
['Text', 'mining', 'is', 'to', 'identify', 'useful', 'information
', '.']
['Current', 'NLP', 'models', 'is', "n't", 'able', 'to', 'solve', '
NLU', 'perfectly', '.']
```

In [7]:
```
# Other examples:
# 1. Possessive cases: Apostrophe (isn't, I've, ...)
tokens = tokenize("Bob's text mining skills are perfect.")
print(tokens)
# 2. Parentheses
tokens = tokenize("Bob's text mining skills (or, NLP) are perfect.")
print(tokens)
# 3. ellipsis
tokens = tokenize("Bob's text mining skills are perfect...")
print(tokens)
```

```
['Bob', "'s", 'text', 'mining', 'skills', 'are', 'perfect', '.']
['Bob', "'s", 'text', 'mining', 'skills', '(', 'or', ',', 'NLP',
')', 'are', 'perfect', '.']
['Bob', "'s", 'text', 'mining', 'skills', 'are', 'perfect', '...']
```

2. Stemming and lemmatization

(https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html (https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html))

Stemming: chops off the ends of words to acquire the root, and often includes the removal of derivational affixes.

e.g., gone -> go, wanted -> want, trees -> tree.

Lemmatization: doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma .

Differences: The two may also differ in that stemming most commonly collapses derivationally related words, whereas lemmatization commonly only collapses the different inflectional forms of a lemma (focus on the concrete semantic meaning).

E.g.: useful -> use(stemming), useful(lemmatization)

PorterStemmer:

Rule-based methods. E.g., SSES->SS, IES->I, NOUNS->NOUN. # misses->miss, flies->fli.

Doc: https://www.nltk.org/api/nltk.stem.html (https://www.nltk.org/api/nltk.stem.html)

In [8]:
```
from nltk.stem import PorterStemmer
ps = PorterStemmer()

def stem(tokens):
    """
    :param tokens: a list of tokens, type: list
    return a list of stemmed words, type: list
    e.g.
    Input: ['Text', 'mining', 'is', 'to', 'identify', 'useful', 'inf
ormation', '.']
    Output: ['text', 'mine', 'is', 'to', 'identifi', 'use', 'inform
', '.']
    """
    ### equivalent code
    # results = list()
    # for token in tokens:
    #     results.append(ps.stem(token))
    # return results

    return [ps.stem(token) for token in tokens]
```

In [9]:
```
tokens = stem(tokenize("Text mining is to identify useful informatio
n."))
print(tokens)
```

```
['text', 'mine', 'is', 'to', 'identifi', 'use', 'inform', '.']
```

In [10]:
```
from nltk.stem import WordNetLemmatizer
lm = WordNetLemmatizer()
def lemmatize(tokens):
    return [lm.lemmatize(token) for token in tokens]
```

In [11]:
```
tokens = lemmatize(tokenize("Text mining is to identify useful infor
mation."))
print(tokens)
```

```
['Text', 'mining', 'is', 'to', 'identify', 'useful', 'information
', '.']
```

3. WordNet

https://www.nltk.org/howto/wordnet.html (https://www.nltk.org/howto/wordnet.html)

- a semantically-oriented dictionary of English,
- similar to a traditional thesaurus but with a richer structure

In [12]:
```
from nltk.corpus import wordnet as wn
```

3.1 synsets

A set of one or more **synonyms** that are interchangeable in some context without changing the truth value of the proposition in which they are embedded.

In [13]:
```
# Look up a word using synsets();
wn.synsets('dog')
```

Out[13]:
```
[Synset('dog.n.01'),
 Synset('frump.n.01'),
 Synset('dog.n.03'),
 Synset('cad.n.01'),
 Synset('frank.n.02'),
 Synset('pawl.n.01'),
 Synset('andiron.n.01'),
 Synset('chase.v.01')]
```

In [14]:
```
wn.synsets('bank')
```

Out[14]:
```
[Synset('bank.n.01'),
 Synset('depository_financial_institution.n.01'),
 Synset('bank.n.03'),
 Synset('bank.n.04'),
 Synset('bank.n.05'),
 Synset('bank.n.06'),
 Synset('bank.n.07'),
 Synset('savings_bank.n.02'),
 Synset('bank.n.09'),
 Synset('bank.n.10'),
 Synset('bank.v.01'),
 Synset('bank.v.02'),
 Synset('bank.v.03'),
 Synset('bank.v.04'),
 Synset('bank.v.05'),
 Synset('deposit.v.02'),
 Synset('bank.v.07'),
 Synset('trust.v.01')]
```

In [15]:
```
print("synset","\t","definition")
for synset in wn.synsets('bank'):
    print(synset, '\t', synset.definition())
```

```
synset   definition
Synset('bank.n.01')      sloping land (especially the slope beside
a body of water)
Synset('depository_financial_institution.n.01')          a financi
al institution that accepts deposits and channels the money into l
ending activities
Synset('bank.n.03')      a long ridge or pile
Synset('bank.n.04')      an arrangement of similar objects in a ro
w or in tiers
Synset('bank.n.05')      a supply or stock held in reserve for fut
ure use (especially in emergencies)
Synset('bank.n.06')      the funds held by a gambling house or the
dealer in some gambling games
Synset('bank.n.07')      a slope in the turn of a road or track; t
he outside is higher than the inside in order to reduce the effect
s of centrifugal force
Synset('savings_bank.n.02')      a container (usually with a slot
in the top) for keeping money at home
Synset('bank.n.09')      a building in which the business of banki
ng transacted
Synset('bank.n.10')      a flight maneuver; aircraft tips laterall
y about its longitudinal axis (especially in turning)
Synset('bank.v.01')      tip laterally
Synset('bank.v.02')      enclose with a bank
Synset('bank.v.03')      do business with a bank or keep an accoun
t at a bank
Synset('bank.v.04')      act as the banker in a game or in gamblin
g
Synset('bank.v.05')      be in the banking business
Synset('deposit.v.02')   put into a bank account
Synset('bank.v.07')      cover with ashes so to control the rate o
f burning
Synset('trust.v.01')     have confidence or faith in
```

In [16]:
```
# this function has an optional pos argument which lets you constrai
n the part of speech of the word:
# pos: part-of-speech
wn.synsets('bank', pos=wn.NOUN)
```

Out[16]:
```
[Synset('bank.n.01'),
 Synset('depository_financial_institution.n.01'),
 Synset('bank.n.03'),
 Synset('bank.n.04'),
 Synset('bank.n.05'),
 Synset('bank.n.06'),
 Synset('bank.n.07'),
 Synset('savings_bank.n.02'),
 Synset('bank.n.09'),
 Synset('bank.n.10')]
```

In [17]:
```
wn.synset('dog.n.01')
```

Out[17]:
```
Synset('dog.n.01')
```

In [18]:
```
print(wn.synset('dog.n.01').definition())
```

```
a member of the genus Canis (probably descended from the common wo
lf) that has been domesticated by man since prehistoric times; occ
urs in many breeds
```

In [19]:
```
wn.synset('dog.n.01').examples()
```

Out[19]:
```
['the dog barked all night']
```

In [20]:
```
wn.synset('dog.n.01').lemma_names()
```

Out[20]:
```
['dog', 'domestic_dog', 'Canis_familiaris']
```

In [ ]:
```
dir(wn.synset('dog.n.01'))
# isA: hyponyms, hypernyms
# part_of: member_holonyms, substance_holonyms, part_holonyms
# being part of: member_meronyms, substance_meronyms, part_meronyms
# domains: topic_domains, region_domains, usage_domains
# attribute: attributes
# entailments: entailments
# causes: causes
# also_sees: also_sees
# verb_groups: verb_groups
# similar_to: similar_tos
```

Check more relations in [http://www.nltk.org/api/nltk.corpus.reader.html?highlight=wordnet (http://www.nltk.org/api/nltk.corpus.reader.html?highlight=wordnet)](http://www.nltk.org/api/nltk.corpus.reader.html?highlight=wordnet)

In [21]:
```
# hypernyms: abstraction
# hyponyms: instantiation

dog = wn.synset('dog.n.01')
print("hypernyms:", dog.hypernyms())
print("hyponyms:", dog.hyponyms())
```

```
hypernyms: [Synset('canine.n.02'), Synset('domestic_animal.n.01')]
hyponyms: [Synset('basenji.n.01'), Synset('corgi.n.01'), Synset('c
ur.n.01'), Synset('dalmatian.n.02'), Synset('great_pyrenees.n.01
'), Synset('griffon.n.02'), Synset('hunting_dog.n.01'), Synset('la
pdog.n.01'), Synset('leonberg.n.01'), Synset('mexican_hairless.n.0
1'), Synset('newfoundland.n.01'), Synset('pooch.n.01'), Synset('po
odle.n.01'), Synset('pug.n.01'), Synset('puppy.n.01'), Synset('spi
tz.n.01'), Synset('toy_dog.n.01'), Synset('working_dog.n.01')]
```

In [22]:
```
print(dog.hypernyms()[0].hypernyms()) # the hypernym of canine
# animals that feeds on flesh
print(dog.hypernyms()[0].hypernyms()[0].hypernyms()) # the hypernym
of carnivore
# placental mammals
print(dog.hypernyms()[0].hypernyms()[0].hypernyms()[0].hypernyms())
# the hypernym of placental
# mammals
# ...
print("root hypernyms for dog:", dog.root_hypernyms())
```

```
[Synset('carnivore.n.01')]
[Synset('placental.n.01')]
[Synset('mammal.n.01')]
root hypernyms for dog: [Synset('entity.n.01')]
```

In [23]:
```
# find common hypernyms
print("root hypernyms for cat:", wn.synset('cat.n.01').hypernyms())
print("root hypernyms for cat:", wn.synset('cat.n.01').root_hypernym
s())
print("the lowest common hypernyms of dog and cat")
print(wn.synset('dog.n.01').lowest_common_hypernyms(wn.synset('cat.
n.01')))
```

```
root hypernyms for cat: [Synset('feline.n.01')]
root hypernyms for cat: [Synset('entity.n.01')]
the lowest common hypernyms of dog and cat
[Synset('carnivore.n.01')]
```

3.2 Similarity

In [24]:
```
dog = wn.synset('dog.n.01')
corgi = wn.synset('corgi.n.01')
bensenji = wn.synset('basenji.n.01')
cat = wn.synset('cat.n.01')
```

In [25]:
```
dog.path_similarity(cat) # dog <- canine <- carnivore -> feline -> c
at
```

Out[25]: 0.2

In [26]:
```
dog.path_similarity(corgi) # corgi <- dog
```

Out[26]: 0.5

In [27]:
```
corgi.path_similarity(bensenji) # bensenji <- dog -> corgi
```

Out[27]: 0.3333333333333333

In [28]:
```
hit = wn.synset('hit.v.01')
slap = wn.synset('slap.v.01')
jump = wn.synset('jump.v.01')
run = wn.synset('run.v.01')
```

In [29]:
```
hit.path_similarity(slap) # 1/7
```

Out[29]:
```
0.14285714285714285
```

In [30]:
```
hit.path_similarity(jump) # 1/6
```

Out[30]:
```
0.16666666666666666
```

also check:

- wup_similarity
- lch_similarity
- res_similarity ...

Find more on https://www.nltk.org/howto/wordnet.html (https://www.nltk.org/howto/wordnet.html)

3.3 Traverse the synsets to build a graph

In [31]:
```
wn_graph_hypernyms = {}
# or you could use networkx package

for synset in list(wn.all_synsets('n'))[:10]:
    for hyp_syn in synset.hypernyms():
        wn_graph_hypernyms[synset.name()] = {**wn_graph_hypernyms.ge
t(synset.name(), {}), **{hyp_syn.name():True}}
```

In [32]:
```
wn_graph_hypernyms['physical_entity.n.01']['entity.n.01']
```

Out[32]:
```
True
```

4. Tips to the assignments

Some corpus in the NLTK.

Reference: https://www.nltk.org/book/ch02.html (https://www.nltk.org/book/ch02.html). You could search for `gutenberg` and `brown` for detailed documentations.

4.1 gutenberg corpus

In [33]:
```
from nltk.corpus import gutenberg as gb
nltk.download("gutenberg")
```

```
[nltk_data] Downloading package gutenberg to
[nltk_data]     /Users/tianqing/nltk_data...
[nltk_data]   Package gutenberg is already up-to-date!
```

Out[33]:
```
True
```

In [34]:
```
file_id = 'austen-sense.txt'
word_list = gb.words(file_id)
```

In [35]:
```
print(word_list[:100])
```

```
['[', 'Sense', 'and', 'Sensibility', 'by', 'Jane', 'Austen', '1811
', ']', 'CHAPTER', '1', 'The', 'family', 'of', 'Dashwood', 'had',
'long', 'been', 'settled', 'in', 'Sussex', '.', 'Their', 'estate',
'was', 'large', ',', 'and', 'their', 'residence', 'was', 'at', 'No
rland', 'Park', ',', 'in', 'the', 'centre', 'of', 'their', 'proper
ty', ',', 'where', ',', 'for', 'many', 'generations', ',', 'they',
'had', 'lived', 'in', 'so', 'respectable', 'a', 'manner', 'as', 't
o', 'engage', 'the', 'general', 'good', 'opinion', 'of', 'their',
'surrounding', 'acquaintance', '.', 'The', 'late', 'owner', 'of',
'this', 'estate', 'was', 'a', 'single', 'man', ',', 'who', 'lived
', 'to', 'a', 'very', 'advanced', 'age', ',', 'and', 'who', 'for',
'many', 'years', 'of', 'his', 'life', ',', 'had', 'a', 'constant',
'companion']
```

In [36]:
```
sents = gb.sents(file_id)
```

In [37]:
```
sents[0]
```

Out[37]:
```
['[', 'Sense', 'and', 'Sensibility', 'by', 'Jane', 'Austen', '1811
', ']']
```

4.2 brown corpus

In [38]:
```
from nltk.corpus import brown
nltk.download("brown")
print(brown.categories())

romance_word_list = brown.words(categories='romance')
```

```
['adventure', 'belles_lettres', 'editorial', 'fiction', 'governmen
t', 'hobbies', 'humor', 'learned', 'lore', 'mystery', 'news', 'rel
igion', 'reviews', 'romance', 'science_fiction']

[nltk_data] Downloading package brown to /Users/tianqing/nltk_dat
a...
[nltk_data]   Package brown is already up-to-date!
```

In [39]:
```
romance_word_list
```

Out[39]:
```
['They', 'neither', 'liked', 'nor', 'disliked', 'the', ...]
```

In [ ]:
```

```