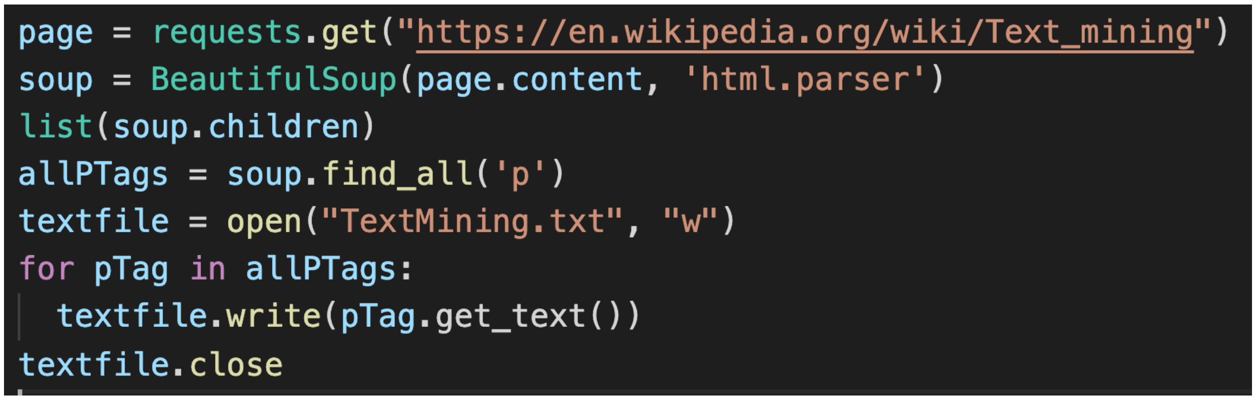
Text mining Assignment

Shirin Mohebbi

1&2. Extract Content:

For this assignment, I choose a Wikipedia web page about text mining! I use a crawler to read content from this web page and then I extract content from its HTML text. I use the BeautifulSoup library to extract content in the HTML p tags. Then I write extracted paragraphs in a file entitled TextMining.



```python
page = requests.get("https://en.wikipedia.org/wiki/Text_mining")
soup = BeautifulSoup(page.content, 'html.parser')
list(soup.children)
allPTags = soup.find_all('p')
textfile = open("TextMining.txt", "w")
for pTag in allPTags:
    textfile.write(pTag.get_text())
textfile.close
```

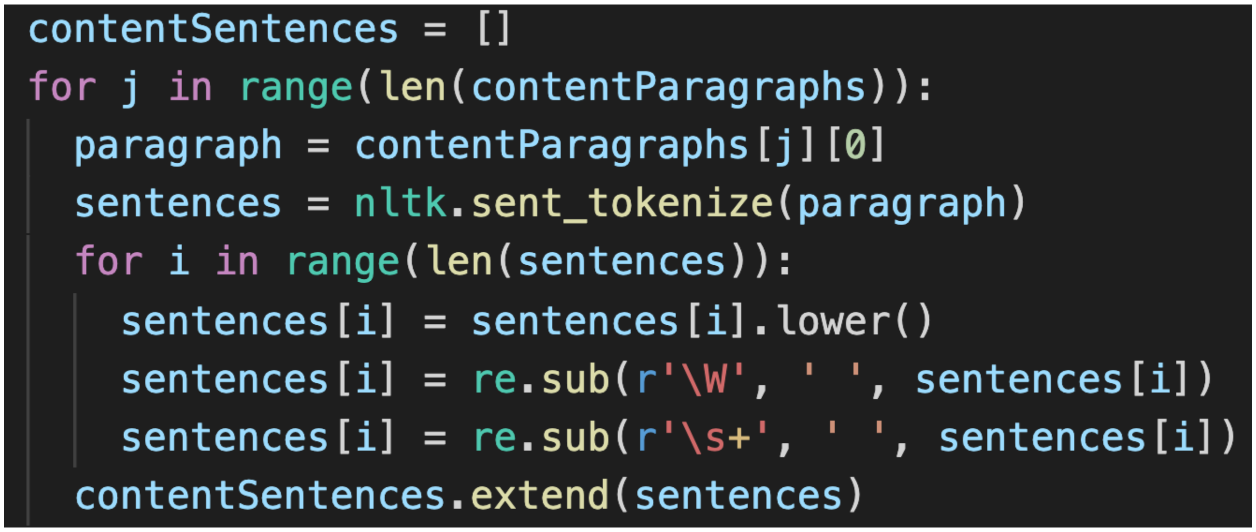
3. Preprocessing:

-Sentence tokenization:

The first step in text analysis and processing is to split the text into sentences and words, a process called tokenization. Tokenizing a text makes further analysis easier. Almost all text analysis applications start with this step.

-Convert text to lower case

-Remove non-word characters. Remove all punctuations

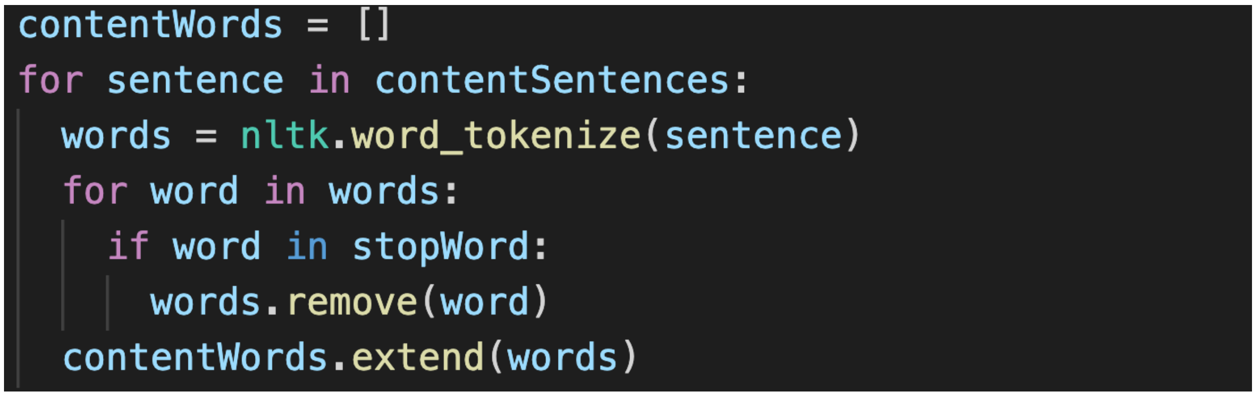


```
contentSentences = []
for j in range(len(contentParagraphs)):
    paragraph = contentParagraphs[j][0]
    sentences = nltk.sent_tokenize(paragraph)
    for i in range(len(sentences)):
        sentences[i] = sentences[i].lower()
        sentences[i] = re.sub(r'\W', ' ', sentences[i])
        sentences[i] = re.sub(r'\s+', ' ', sentences[i])
    contentSentences.extend(sentences)
```

-Word tokenization:

-Removing stopwords:

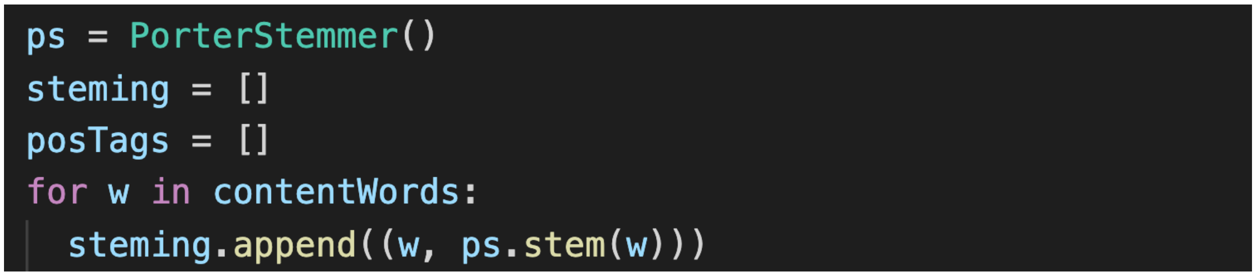
A language like English has many "fluff" words (technically called "stopwords") that are necessary for speech and writing but do not carry value in analysis. NLTK can identify and remove these stopwords to help text processing focus on requisite words.



```
contentWords = []
for sentence in contentSentences:
  words = nltk.word_tokenize(sentence)
  for word in words:
    if word in stopWord:
      words.remove(word)
  contentWords.extend(words)
```

-Stemming:

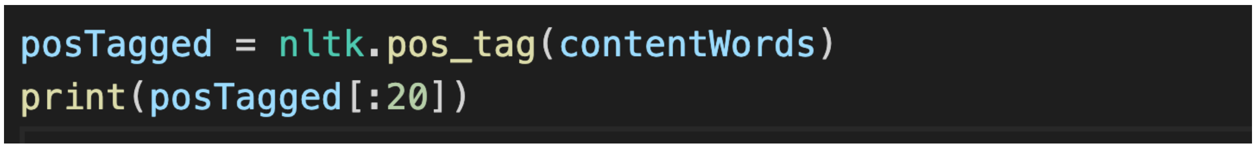
Words are typically inflected (e.g., letters suffixed, affixed, etc.) to express their forms (e.g., plural, tense, etc.). Dog -> Dogs is an example of inflection. Usually, words must be compared in their native forms for effective text matching. Stemming is a method to convert a word to a non-infected form. The essence of stemming is the same: to reduce a word to its most native form. But they differ in how they do it.



```
ps = PorterStemmer()
steming = []
posTags = []
for w in contentWords:
  steming.append((w, ps.stem(w)))
```

-Part of speech tagging:

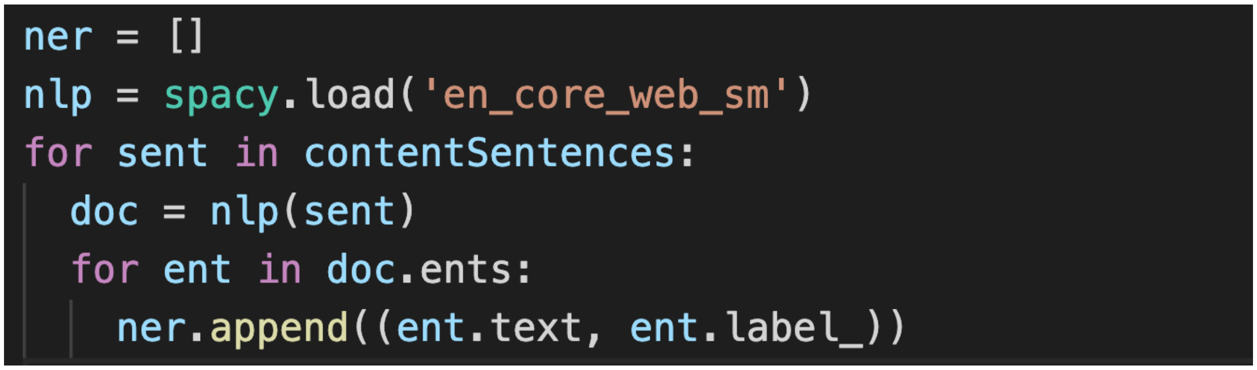
NLTK can identify words' parts of speech (POS). Identifying POS is necessary, as a word has different meanings in different contexts. The word "code" as a noun could mean "a system of words for secrecy" or "program instructions," and a verb, it could mean "convert a message into secret form" or "write instructions for a computer." This context cognizance is necessary for correct text comprehension.



```
posTagged = nltk.pos_tag(contentWords)
print(posTagged[:20])
```

-Named Entity Recognition:

NER is a standard NLP problem that involves spotting named entities (people, places, organizations, etc.) from a chunk of text, and classifying them into a predefined set of categories. spaCy is regarded as the fastest NLP framework in Python, with single optimized functions for each of the NLP tasks it implements. Being easy to learn and use, one can easily perform simple tasks using a few lines of code.



```
ner = []
nlp = spacy.load('en_core_web_sm')
for sent in contentSentences:
  doc = nlp(sent)
  for ent in doc.ents:
    ner.append((ent.text, ent.label_))
```

-Parse tree:

The parse tree breaks down the sentence into structured parts so that the computer can easily understand and process it. In order for the parsing algorithm to construct this parse tree, a set of rewrite rules, which describe what tree structures are legal, need to be constructed. To derive parse tree i’ve used RerankingParser library. It gets sentence and return the best parse tree for that sentence.

```python
model_dir = find('models/bllip_wsj_no_aux').path
parser = RerankingParser.from_unified_model_dir(model_dir)
best = parser.parse(contentSentences[0])
print(contentSentences[0],'\n')
print(best.get_parser_best(),'\n')
```

4.WordNet for verbs:

WordNet is a lexical database that has been used by a major search engine. From WordNet, information about a given word or phrase can be calculated. To use wordnet for verbs, we should first find all verbs, we search through a pos tagged array and choose all words whose tag starts with “VB”.

```python
verbs = []
for (word,tag) in posTagged:
    if tag[:2] == 'VB':
        verbs.append(word)
```

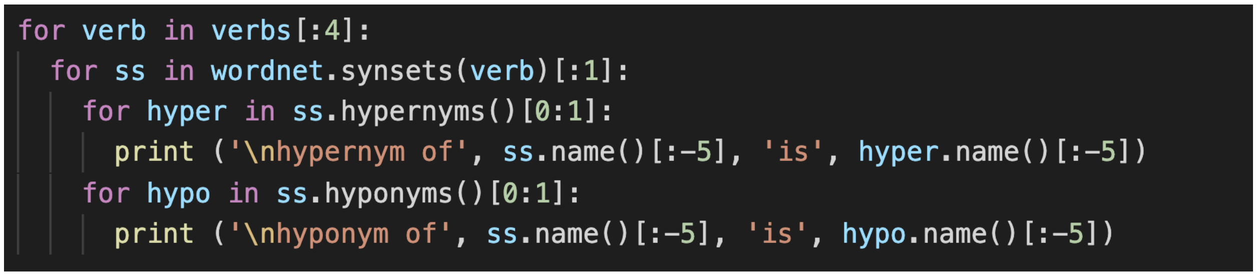
a. Finding the synonym of each word:

wordnet can send the synonyms and antonyms of a given word. the lemmas will be synonyms.

```python
verbsSynoyms = []
for word in verbs:
    syn = []
    for synset in wordnet.synsets(word):
        for lemma in synset.lemmas():
            syn.append(lemma.name())
    verbsSynoyms.append((word, syn))
```

b. Hypernym and Hyponym relations:

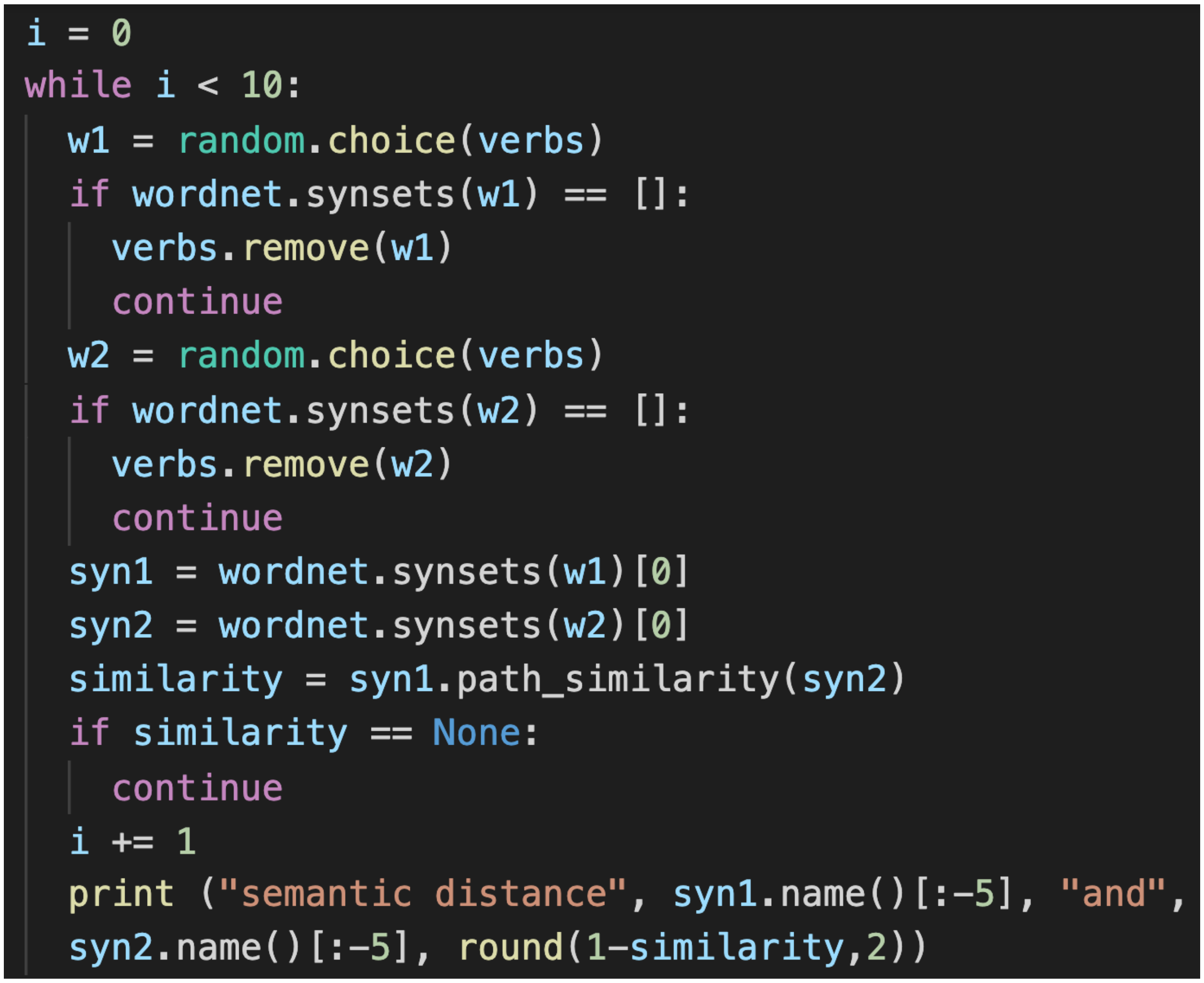
Hyponyms and Hypernyms are specific and generalized concepts respectively. For example, ‘beach house’ and ‘guest house’ are hyponyms of ‘house’. They are more specific concepts of ‘house’. And ‘house’ is a hypernym of ‘guest house’ because it is a general concept. ‘Pasta’ is a hyponym of ‘dish’ and ‘dish’ is a hypernym of ‘pasta’.



```python
for verb in verbs[:4]:
  for ss in wordnet.synsets(verb)[:1]:
    for hyper in ss.hypernyms()[0:1]:
      print ('\nhypernym of', ss.name()[:-5], 'is', hyper.name()[:-5])
    for hypo in ss.hyponyms()[0:1]:
      print ('\nhyponym of', ss.name()[:-5], 'is', hypo.name()[:-5])
```

c. Semantic Distance:

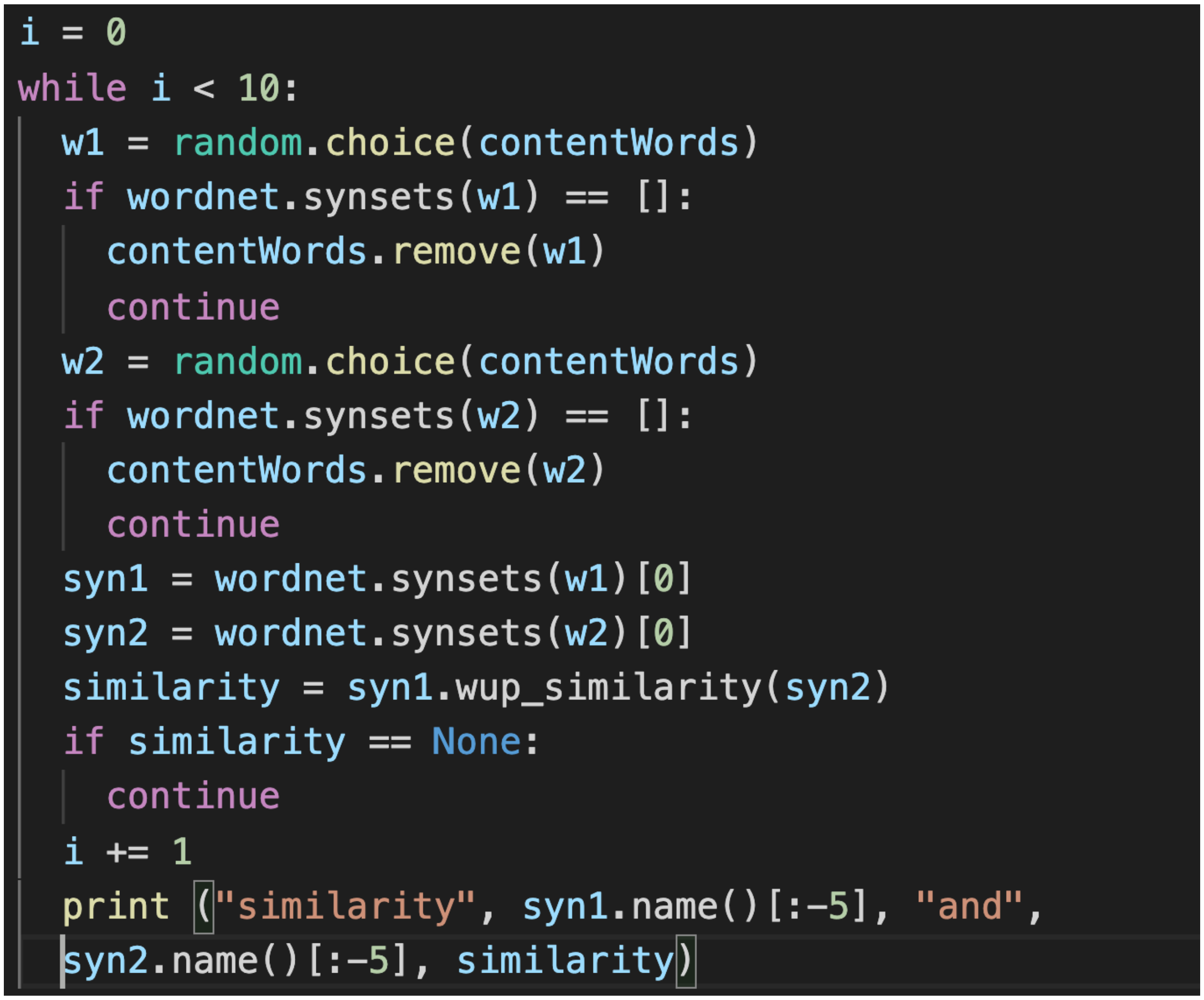
I used path similarity. This metric returns a score denoting how similar two-word senses are, based on the shortest path that connects the senses in the is-a (hypernym/hypnoym) taxonomy. The score is in the range of 0 to 1. Because its value represent similarity i used 1 - similarity as distance.



```python
i = 0
while i < 10:
  w1 = random.choice(verbs)
  if wordnet.synsets(w1) == []:
    verbs.remove(w1)
    continue
  w2 = random.choice(verbs)
  if wordnet.synsets(w2) == []:
    verbs.remove(w2)
    continue
  syn1 = wordnet.synsets(w1)[0]
  syn2 = wordnet.synsets(w2)[0]
  similarity = syn1.path_similarity(syn2)
  if similarity == None:
    continue
  i += 1
  print ("semantic distance", syn1.name()[:-5], "and",
  syn2.name()[:-5], round(1-similarity,2))
```

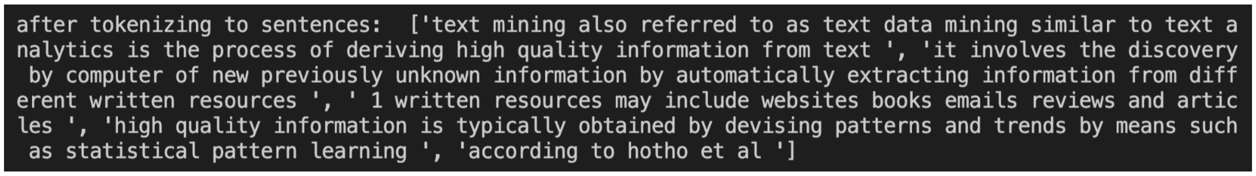
5. Similarity:

The similarity is a task in the area of Natural Language Processing (NLP) that scores the relationship between words using a defined metric. I used Wu-Palmer. Wu-Palmer Similarity Returns a score denoting how similar two-word senses are, based on the depth of the two senses in the taxonomy and that of their Least Common Subsumer (most specific ancestor node).

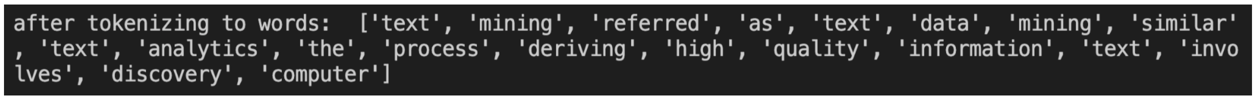


```python
i = 0
while i < 10:
    w1 = random.choice(contentWords)
    if wordnet.synsets(w1) == []:
        contentWords.remove(w1)
        continue
    w2 = random.choice(contentWords)
    if wordnet.synsets(w2) == []:
        contentWords.remove(w2)
        continue
    syn1 = wordnet.synsets(w1)[0]
    syn2 = wordnet.synsets(w2)[0]
    similarity = syn1.wup_similarity(syn2)
    if similarity == None:
        continue
    i += 1
    print ("similarity", syn1.name()[:-5], "and",
    syn2.name()[:-5], similarity)
```

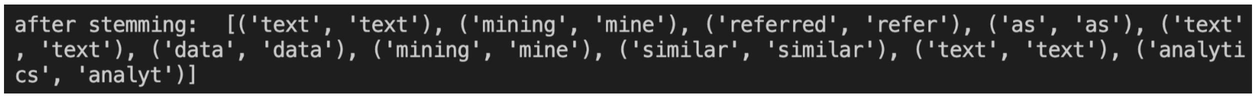
Results:



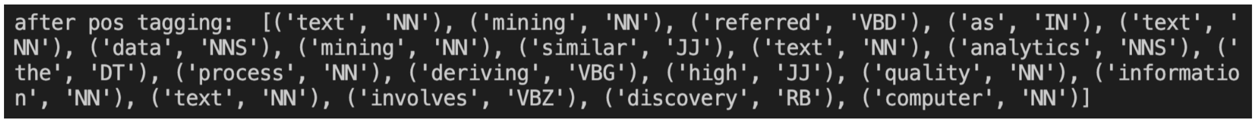
```
after tokenizing to sentences:  ['text mining also referred to as text data mining similar to text a
nalytics is the process of deriving high quality information from text ', 'it involves the discovery
 by computer of new previously unknown information by automatically extracting information from diff
erent written resources ', ' 1 written resources may include websites books emails reviews and artic
les ', 'high quality information is typically obtained by devising patterns and trends by means such
 as statistical pattern learning ', 'according to hotho et al ']
```



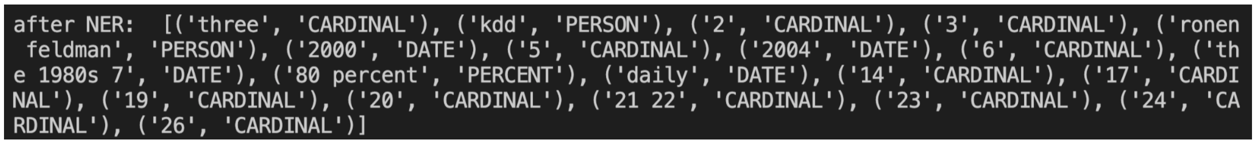
```
after tokenizing to words:  ['text', 'mining', 'referred', 'as', 'text', 'data', 'mining', 'similar'
, 'text', 'analytics', 'the', 'process', 'deriving', 'high', 'quality', 'information', 'text', 'invo
lves', 'discovery', 'computer']
```



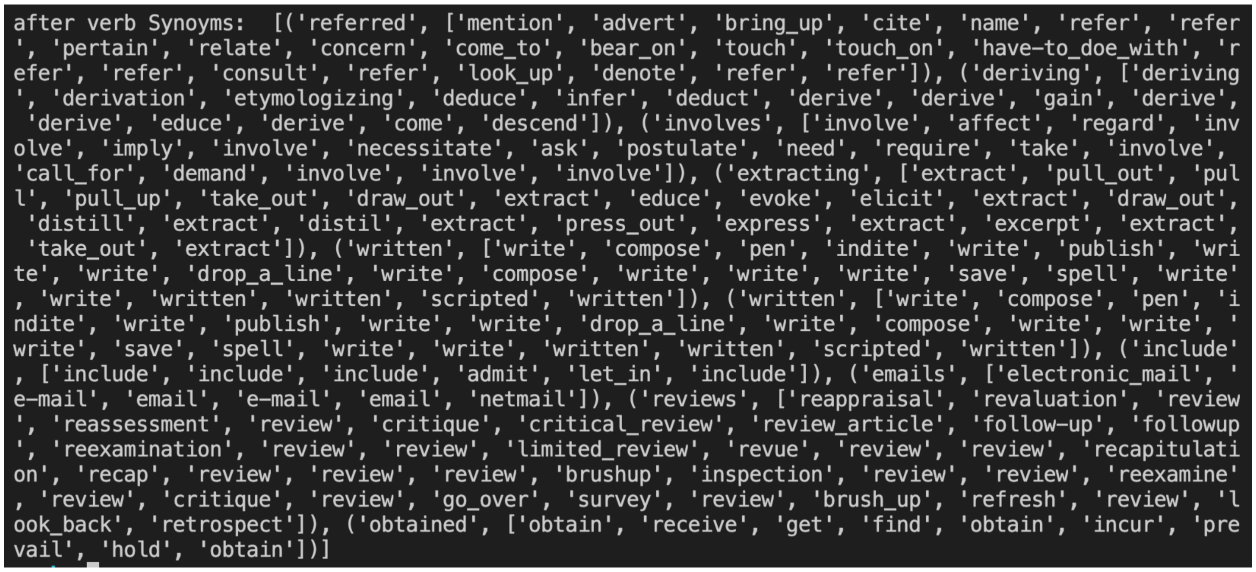
```
after stemming:  [('text', 'text'), ('mining', 'mine'), ('referred', 'refer'), ('as', 'as'), ('text'
, 'text'), ('data', 'data'), ('mining', 'mine'), ('similar', 'similar'), ('text', 'text'), ('analyti
cs', 'analyt')]
```



```
after pos tagging:  [('text', 'NN'), ('mining', 'NN'), ('referred', 'VBD'), ('as', 'IN'), ('text', '
NN'), ('data', 'NNS'), ('mining', 'NN'), ('similar', 'JJ'), ('text', 'NN'), ('analytics', 'NNS'), ('
the', 'DT'), ('process', 'NN'), ('deriving', 'VBG'), ('high', 'JJ'), ('quality', 'NN'), ('informatio
n', 'NN'), ('text', 'NN'), ('involves', 'VBZ'), ('discovery', 'RB'), ('computer', 'NN')]
```

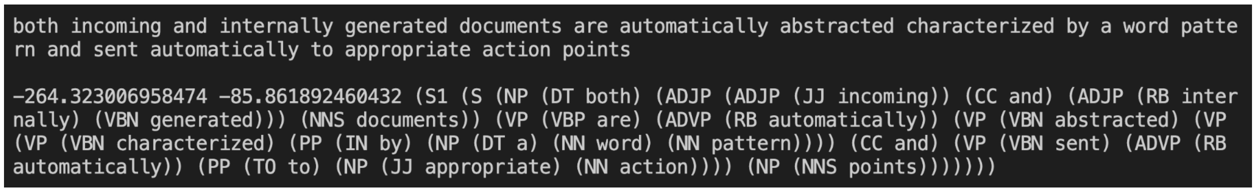


```
after NER:  [('three', 'CARDINAL'), ('kdd', 'PERSON'), ('2', 'CARDINAL'), ('3', 'CARDINAL'), ('ronen
 feldman', 'PERSON'), ('2000', 'DATE'), ('5', 'CARDINAL'), ('2004', 'DATE'), ('6', 'CARDINAL'), ('th
e 1980s 7', 'DATE'), ('80 percent', 'PERCENT'), ('daily', 'DATE'), ('14', 'CARDINAL'), ('17', 'CARDI
NAL'), ('19', 'CARDINAL'), ('20', 'CARDINAL'), ('21 22', 'CARDINAL'), ('23', 'CARDINAL'), ('24', 'CA
RDINAL'), ('26', 'CARDINAL')]
```



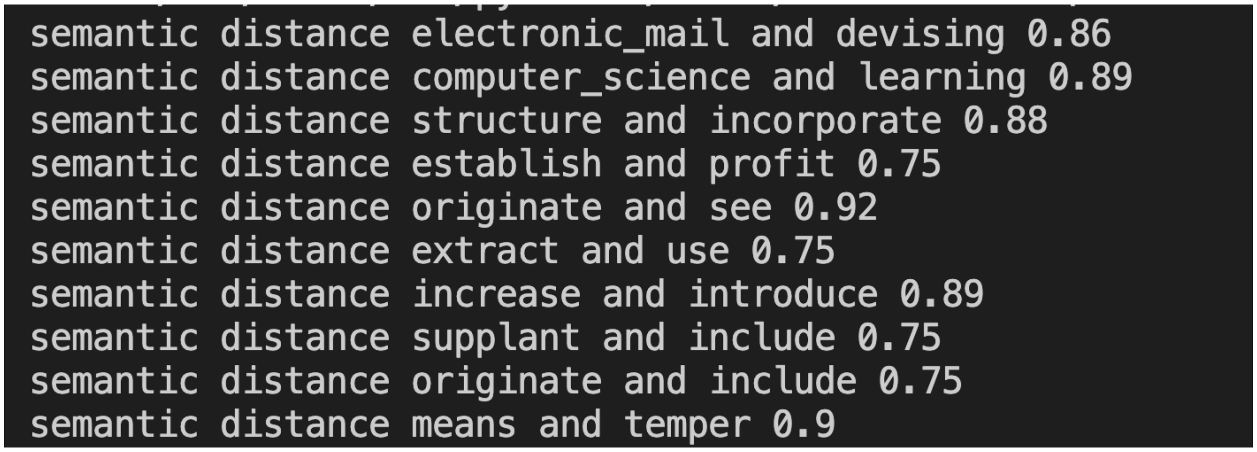
```
after verb Synoyms:  [('referred', ['mention', 'advert', 'bring_up', 'cite', 'name', 'refer', 'refer
', 'pertain', 'relate', 'concern', 'come_to', 'bear_on', 'touch', 'touch_on', 'have-to_doe_with', 'r
efer', 'refer', 'consult', 'refer', 'look_up', 'denote', 'refer', 'refer']), ('deriving', ['deriving
', 'derivation', 'etymologizing', 'deduce', 'infer', 'deduct', 'derive', 'derive', 'gain', 'derive',
 'derive', 'educe', 'derive', 'come', 'descend']), ('involves', ['involve', 'affect', 'regard', 'inv
olve', 'imply', 'involve', 'necessitate', 'ask', 'postulate', 'need', 'require', 'take', 'involve',
'call_for', 'demand', 'involve', 'involve', 'involve']), ('extracting', ['extract', 'pull_out', 'pul
l', 'pull_up', 'take_out', 'draw_out', 'extract', 'educe', 'evoke', 'elicit', 'extract', 'draw_out',
 'distill', 'extract', 'distil', 'extract', 'press_out', 'express', 'extract', 'excerpt', 'extract',
 'take_out', 'extract']), ('written', ['write', 'compose', 'pen', 'indite', 'write', 'publish', 'wri
te', 'write', 'drop_a_line', 'write', 'compose', 'write', 'write', 'write', 'save', 'spell', 'write'
, 'write', 'written', 'written', 'scripted', 'written']), ('written', ['write', 'compose', 'pen', 'i
ndite', 'write', 'publish', 'write', 'write', 'drop_a_line', 'write', 'compose', 'write', 'write', '
write', 'save', 'spell', 'write', 'write', 'written', 'written', 'scripted', 'written']), ('include'
, ['include', 'include', 'include', 'admit', 'let_in', 'include']), ('emails', ['electronic_mail', '
e-mail', 'email', 'e-mail', 'email', 'netmail']), ('reviews', ['reappraisal', 'revaluation', 'review
', 'reassessment', 'review', 'critique', 'critical_review', 'review_article', 'follow-up', 'followup
', 'reexamination', 'review', 'review', 'limited_review', 'revue', 'review', 'review', 'recapitulati
on', 'recap', 'review', 'review', 'review', 'brushup', 'inspection', 'review', 'review', 'reexamine'
, 'review', 'critique', 'review', 'go_over', 'survey', 'review', 'brush_up', 'refresh', 'review', 'l
ook_back', 'retrospect']), ('obtained', ['obtain', 'receive', 'get', 'find', 'obtain', 'incur', 'pre
vail', 'hold', 'obtain'])]
```

Parse tree:

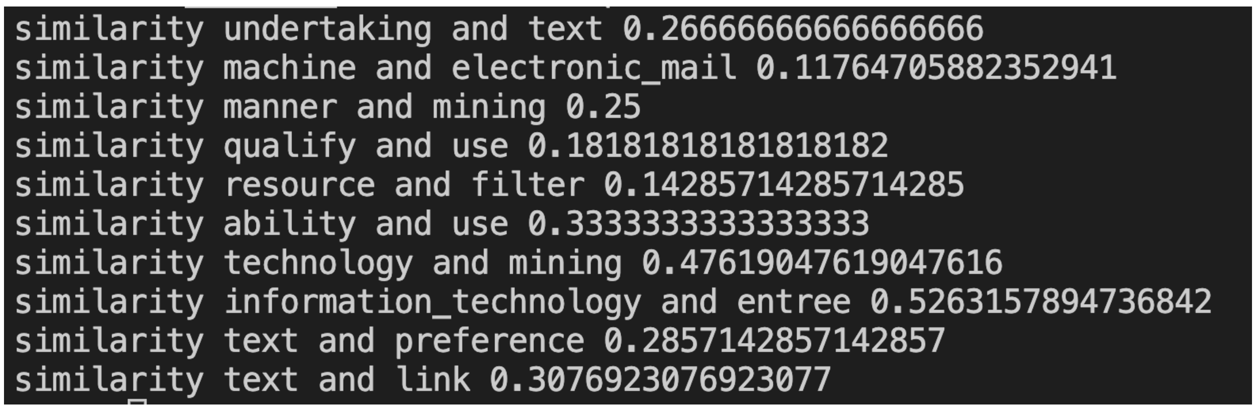


```
both incoming and internally generated documents are automatically abstracted characterized by a word patte
rn and sent automatically to appropriate action points

-264.323006958474 -85.861892460432 (S1 (S (NP (DT both) (ADJP (ADJP (JJ incoming)) (CC and) (ADJP (RB inter
nally) (VBN generated))) (NNS documents)) (VP (VBP are) (ADVP (RB automatically)) (VP (VBN abstracted) (VP
(VP (VBN characterized) (PP (IN by) (NP (DT a) (NN word) (NN pattern)))) (CC and) (VP (VBN sent) (ADVP (RB
automatically)) (PP (TO to) (NP (JJ appropriate) (NN action)))) (NP (NNS points)))))))
```



```
semantic distance electronic_mail and devising 0.86
semantic distance computer_science and learning 0.89
semantic distance structure and incorporate 0.88
semantic distance establish and profit 0.75
semantic distance originate and see 0.92
semantic distance extract and use 0.75
semantic distance increase and introduce 0.89
semantic distance supplant and include 0.75
semantic distance originate and include 0.75
semantic distance means and temper 0.9
```


```
similarity undertaking and text 0.26666666666666666
similarity machine and electronic_mail 0.11764705882352941
similarity manner and mining 0.25
similarity qualify and use 0.18181818181818182
similarity resource and filter 0.14285714285714285
similarity ability and use 0.3333333333333333
similarity technology and mining 0.47619047619047616
similarity information_technology and entree 0.5263157894736842
similarity text and preference 0.2857142857142857
similarity text and link 0.3076923076923077
```