Keywords

June 20, 2024

```
<h1>Natural Language Processing</h1>
assignment 6
<h3>General Information:</h3>
<p>Please do not add or delete any cells. Answers belong into the corresponding cells (below tl
<p>Code cells where you are supposed to give your answer often include the line  ```raise NotIn
<h3>Submission:</h3>
<p>Please submit your notebook via the web interface (in the main view -> Assignments -> Submit
<h3>Group Work:</h3>
<p>You are allowed to work in groups of up to two people. Please enter the UID (your username h
<h3>Questions about the Assignment:</h3>
<p>If you have questions about the assignment please post them in the LEA forum before the dead
```

[1]:

```python
'''
Group Work:
Enter the username of each team member into the variables.
If you work alone please leave the second variable empty.
'''
member1 = 'mfarra2s'
member2 = 'rhusai2s'
member3 = ''
```

1 Introduction to spaCy

SpaCy is a tool that does tokenization, parsing, tagging and named entity regocnition (among other things).

When we parse a document via spaCy, we get an object that holds sentences and tokens, as well as their POS tags, dependency relations and so on.

Look at the next cell for an example.

[2]:

```python
import spacy

# Load the English language model
nlp = spacy.load('/srv/shares/NLP/spacy/en_core_web_sm')

# Our sample input
```

```python
text = 'SpaCy is capable of    tagging, parsing and annotating text. It recognizes sentences and stop words.'

# Parse the sample input
doc = nlp(text)

# For every sentence
for sent in doc.sents:
    # For every token
    for token in sent:
        # Print the token itself, the pos tag,
        # dependency tag and whether spacy thinks this is a stop word
        print(token, token.pos_, token.dep_, token.is_stop)

print('-'*30)
print('The nouns and proper nouns in this text are:')
# Print only the nouns:
for token in doc:
    if token.pos_ in ['NOUN', 'PROPN']:
        print(token)
```

```
SpaCy PROPN nsubj False
is AUX ROOT True
capable ADJ acomp False
of ADP prep True
    SPACE dep False
tagging NOUN pobj False
, PUNCT punct False
parsing VERB conj False
and CCONJ cc True
annotating VERB conj False
text NOUN dobj False
. PUNCT punct False
It PRON nsubj True
recognizes VERB ROOT False
sentences NOUN dobj False
and CCONJ cc True
stop VERB conj False
words NOUN dobj False
. PUNCT punct False
------------------------------
The nouns and proper nouns in this text are:
SpaCy
tagging
text
sentences
words
```

1.1 SpaCy A) [5 points]

1.1.1 Splitting text into sentences

You are given the text in the next cell.

```
text = '''
This is a sentence.
Mr. A. said this was another!
But is this a sentence?
The abbreviation Merch. means merchant(s).
At certain univ. in the U.S. and U.K. they study NLP.
'''
```

Use spaCy to split this into sentences. Store the resulting sentences (each as a **single** string) in the list `sentences`. Make sure to convert the tokens to strings (e.g. via str(token)).

[3]:

```
import spacy
nlp = spacy.load('/srv/shares/NLP/spacy/en_core_web_sm')

text = '''
This is a sentence. Mr. A. said this was another!
But is this a sentence? The abbreviation Merch. means merchant(s).
At certain Univ. in the U.S. and U.K. they study NLP.
'''
sentences = []

doc = nlp(text)

# For every sentence
for sent in doc.sents:
    sentences.append(str(sent))

for sentence in sentences:
    print(sentence)
    print('.')
    assert type(sentence) == str, 'You need to convert this to a single string!'
```

```


.
This is a sentence.
.
Mr. A. said this was another!

.
But is this a sentence?
.
The abbreviation Merch. means merchant(s).
```

```
.
At certain Univ.
.
in the U.S. and U.K. they study NLP.

.
```

[4]:

```
# This is a test cell, please ignore it!
```

1.2 SpaCy B) [5 points]

1.2.1 Cluster the text by POS tag

Next we want to cluster the text by the corresponding part-of-speech (POS) tags.

The result should be a dictionary `pos_tags` where the keys are the POS tags and the values are lists of words with those POS tags. Make sure your words are converted to **strings**.

Example:

```
pos_tags['VERB'] # Output: ['said', 'means', 'study']
pos_tags['ADJ']  # Output: ['certain']
...
```

[5]:

```
import spacy
nlp = spacy.load('/srv/shares/NLP/spacy/en_core_web_sm')

text = '''
This is a sentence. Mr. A. said this was another!
But is this a sentence? The abbreviation Merch. means merchant(s).
At certain Univ. in the U.S. and U.K. they study NLP.
'''

pos_tags = dict()

doc = nlp(text)

for sent in doc.sents:
    for token in sent:
        if token.pos_ in pos_tags.keys():
            pos_tags[token.pos_].append(str(token))
        else:
            pos_tags[token.pos_]= []
            pos_tags[token.pos_].append(str(token))


for key in pos_tags:
    print('The words with the POS tag {} are {}.'.format(key, pos_tags[key]))
    for token in pos_tags[key]:
        assert type(token) == str, 'Each token should be a string'
```

```
The words with the POS tag SPACE are ['\n', '\n', '\n', '\n'].
The words with the POS tag PRON are ['This', 'this', 'another', 'this', 'they'].
The words with the POS tag AUX are ['is', 'was', 'is'].
The words with the POS tag DET are ['a', 'a', 'The', 'the'].
The words with the POS tag NOUN are ['sentence', 'sentence', 'abbreviation'].
The words with the POS tag PUNCT are ['.', '!', '?', '.', ')', '.', '.', '.'].
The words with the POS tag PROPN are ['Mr.', 'A.', 'Merch', 'merchant(s',
'Univ', 'U.S.', 'U.K.', 'NLP'].
The words with the POS tag VERB are ['said', 'means', 'study'].
The words with the POS tag CCONJ are ['But', 'and'].
The words with the POS tag ADP are ['At', 'in'].
The words with the POS tag ADJ are ['certain'].
```

[6]:

```python
# This is a test cell, please ignore it!
```

2 SpaCy C) [5 points]

2.0.1 Stop word removal

Stop words are words that appear often in a language and don't hold much meaning for a NLP task. Examples are the words `a, to, the, this, has, ...`. This depends on the task and domain you are working on.

SpaCy has its own internal list of stop words. Use spaCy to remove all stop words from the given text. Store your result as a **single string** in the variable `stopwords_removed`.

[7]:

```python
import spacy
nlp = spacy.load('/srv/shares/NLP/spacy/en_core_web_sm')

text = '''
This is a sentence. Mr. A. said this was another!
But is this a sentence? The abbreviation Merch. means merchant(s).
At certain Univ. in the U.S. and U.K. they study NLP.
'''

stopwords_removed = ''

doc = nlp(text)

for sent in doc.sents:
    for token in sent:
        if not token.is_stop:
            stopwords_removed = stopwords_removed + ' ' + str(token)

print(stopwords_removed)
assert type(stopwords_removed) == str, 'Your answer should be a single string!'
```

```
sentence . Mr. A. said !
sentence ? abbreviation Merch . means merchant(s ) .
certain Univ . U.S. U.K. study NLP .
```

[8]:

```
# This is a test cell, please ignore it!
```

3 SpaCy D) [2 points]

3.0.1 Dependency Tree

We now want to use spaCy to visualize the dependency tree of a certain sentence. Look at the Jupyter Example on the spaCy website. Render the tree.

[9]:

```
import spacy
from spacy import displacy

nlp = spacy.load('/srv/shares/NLP/spacy/en_core_web_sm')

text = 'Dependency Parsing is helpful for many tasks.'

doc = nlp(text)
displacy.render(doc, style="dep")
```

```
<IPython.core.display.HTML object>
```

4 SpaCy E) [5 points]

4.0.1 Dependency Parsing

Use spaCy to extract all subjects and objects from the text. We define a subject as any word that has `subj` in its dependency tag (e.g. `nsubj`, `nsubjpass`, …). Similarly we define an object as any token that has `obj` in its dependency tag (e.g. `dobj`, `pobj`, etc.).

For each sentence extract the subject, root node `ROOT` of the tree and object and store them as a single string in a list. Name this list `subj_obj`.

Example:

“‘ text = ‘Learning multiple ways of representing text is cool. We can access parts of the sentence with dependency tags.’

subj_obj = [‘Learning ways text is’, ‘We access parts sentence tags’]

[10]:

```
import spacy
nlp = spacy.load('/srv/shares/NLP/spacy/en_core_web_sm')

text = '''
This is a sentence. Mr. A. said this was another!
But is this a sentence? The abbreviation Merch. means merchant(s).
```

```
At certain Univ. in the U.S. and U.K. they study NLP.
'''

subj_obj = []

doc = nlp(text)

for sent in doc.sents:
    cleaned_sentence = ''
    for token in sent:
        dep = str(token.dep_)
        if 'subj' in dep or 'ROOT' in dep or 'obj' in dep:
            cleaned_sentence = cleaned_sentence + ' ' + str(token)
    if not len(cleaned_sentence) ==0:
        subj_obj.append(cleaned_sentence)


for cleaned_sent in subj_obj:
    print(cleaned_sent)
    assert type(cleaned_sent) == str, 'Each cleaned sentence should be a string!
 ↪'
```

```
 This is
 A. said this
 is this
 abbreviation means merchant(s
 At Univ
 U.S. they study NLP
```

[11]:

```
# This is a test cell, please ignore it!
```

5 Keyword Extraction

In this assignment we want to write a keyword extractor. There are several methods of which we want to explore a few.

We want to extract keywords from our Yelp reviews.

5.1 POS tag based extraction

When we look at keywords we realize that they are often combinations of nouns and adjectives. The idea is to find all sequences of nouns and adjectives in a corpus and count them. The n most frequent ones are then our keywords.

A keyword (or keyphrase) by this definition is any combination of nouns (NOUN) and adjectives (ADJ) that ends in a noun. We also count proper nouns (PROPN) as nouns.

5.2 POS tag based extraction A) [35 points]

5.2.1 POSKeywordExtractor

Please complete the function `keywords` in the class `POSKeywordExtractor`.

You are given the file `wiki_nlp.txt`, which has the raw text from all top-level Wikipedia pages under the category `Natural language processing`. Use this for extracting your keywords.

Example:

Let us look at the definition of an index term or keyword from Wikipedia. Here I highlighted all combinations of nouns and adjectives that end in a noun. All the highlighted words are potential keywords.

An **index term**, **subject term**, **subject heading**, or **descriptor**, in **information retrieval**, is a **term** that captures the **essence** of the **topic** of a **document**. **Index terms** make up a **controlled vocabulary** for **use** in **bibliographic records**.

Rules:

- A keyphrase is a sequence of nouns, adjectives and proper nouns ending in a noun or proper noun.
- Keywords / Keyphrases **can not go over sentence boundaries**.
- We always take the longest sequence of nouns, adjectives and proper nouns
  - Consider the sentence `She studies natural language processing.`. The only extracted keyphrase here will be `('natural', 'language', 'processing')`.
- Consider the sentence `neural networks massively increased the performance.`:
  - Here our keyphrase would be `neural networks`, not `neural networks massively`.
  - Our keyphrases are always the longest sequence of nouns and adjectives ending in a noun

[12]:
```
%%time
from typing import List, Tuple, Iterable
from collections import Counter
import spacy
from spacy.tokens import Token
import pickle


class POSKeywordExtractor:

    def __init__(self):
        # Set up SpaCy in a more efficient way by disabling what we do not need
        # This is the dependency parser (parser) and the named entity␣
 ↪recognizer (ner)
        self.nlp = spacy.load(
            '/srv/shares/NLP/spacy/en_core_web_sm',
            disable=['ner', 'parser']
        )
        # Add the sentencizer to quickly split our text into sentences
        self.nlp.add_pipe('sentencizer')
        # Increase the maximum length of text SpaCy can parse in one go
        self.nlp.max_length = 1500000

    def validate_keyphrase(self, candidate: Iterable[Token]) -> Iterable[Token]:
        '''
        Takes in a list of tokens which are all proper nouns, nouns or␣
↪adjectives
        and returns the longest sequence that ends in a proper noun or noun

        Args:
            candidate         -- List of spacy tokens
        Returns:
            longest_keyphrase -- The longest sequence that ends in a noun
                                 or proper noun

        Example:
            candidate = [neural, networks, massively]
            longest_keyphrase = [neural, networks]
        '''
        for i in range(len(candidate)-1, -1, -1):
            if candidate[i].pos_ in {'NOUN', 'PROPN'}:
                return candidate[:i+1]
        return []

    def keywords(self, text: str, n_keywords: int, min_words: int) ->␣
↪List[Tuple[Tuple[str], int]]:
        '''
        Extract the top n most frequent keywords from the text.
        Keywords are sequences of adjectives and nouns that end in a noun

        Arguments:
            text       -- the raw text from which to extract keywords
            n_keywords -- the number of keywords to return
            min_words  -- the number of words a potential keyphrase has to␣
↪include
                          if this is set to 2, then only keyphrases consisting␣
↪of 2+ words are counted
        Returns:
            keywords   -- List of keywords and their count, sorted by the count
        '''
        doc = self.nlp(text)
        candidate = []
        keywords = []

        for sent in doc.sents:
            for token in sent:
                if token.pos_ in {'ADJ', 'NOUN', 'PROPN'}:
                    candidate.append(token)
                else:
                    if candidate:
                        validated = self.validate_keyphrase(candidate)
                        if len(validated) >= min_words:
                            keywords.append(tuple(token.text for token in␣
 ↪validated))
                        candidate = []
            # Check for the last candidate at the end of the sentence
            if candidate:
                validated = self.validate_keyphrase(candidate)
                if len(validated) >= min_words:
                    keywords.append(tuple(token.text for token in validated))
                candidate = []


        keyword_counter = Counter(keywords)

        most_common_keywords = keyword_counter.most_common(n_keywords)

        return most_common_keywords


# def process_file_in_chunks(file_path: str, chunk_size: int = 800000) -> str:
#     with open(file_path, 'r') as corpus_file:
#         while True:
#             chunk = corpus_file.read(chunk_size)
#             if not chunk:
#                 break
#             yield chunk

# extractor = POSKeywordExtractor()
# keywords_counter = Counter()

# for chunk in process_file_in_chunks('/srv/shares/NLP/datasets/wiki/wiki_nlp.
 ↪txt'):
#     chunk_keywords = extractor.keywords(chunk.lower(), n_keywords=15,␣
 ↪min_words=1)
#     keywords_counter.update(chunk_keywords)

# for keyword in keywords_counter:
#     print('The keyword {} appears {} times.'.format(*keyword))


with open('/srv/shares/NLP/datasets/wiki/wiki_nlp.txt', 'r') as corpus_file:
    text = corpus_file.read()
```

```
keywords = POSKeywordExtractor().keywords(text.lower(), n_keywords=15,␣
 ↪min_words=1)

'''
Expected output:
The keyword ('words',) appears 353 times.
The keyword ('text',) appears 342 times.
The keyword ('example',) appears 263 times.
The keyword ('word',) appears 231 times.
The keyword ('natural', 'language', 'processing') appears 184 times.
...
'''
for keyword in keywords:
    print('The keyword {} appears {} times.'.format(*keyword))
```

```
The keyword ('words',) appears 355 times.
The keyword ('text',) appears 340 times.
The keyword ('example',) appears 259 times.
The keyword ('word',) appears 241 times.
The keyword ('natural', 'language', 'processing') appears 184 times.
The keyword ('documents',) appears 165 times.
The keyword ('language',) appears 146 times.
The keyword ('information',) appears 137 times.
The keyword ('set',) appears 133 times.
The keyword ('system',) appears 122 times.
The keyword ('systems',) appears 120 times.
The keyword ('t',) appears 120 times.
The keyword ('references',) appears 118 times.
The keyword ('sentence',) appears 116 times.
The keyword ('number',) appears 113 times.
CPU times: user 19.4 s, sys: 3.31 s, total: 22.7 s
Wall time: 22.7 s
```

[13]:

```
# This is a test cell, please ignore it!
```

5.2.2 POS tag based extraction B) [4 points]

Rerun the keyword extrator with a minimum word count of `min_words=2` and a keyword count of `n_keywords=15`.

Store this in the variable `keywords_2`. Print the result.

Make sure to convert the input text to lower case!

[14]:

```
keywords_2 = []

extractor = POSKeywordExtractor()
keywords_counter = Counter()
```

```
for chunk in process_file_in_chunks('/srv/shares/NLP/datasets/wiki/wiki_nlp.txt'):
    chunk_keywords = extractor.keywords(chunk.lower(), n_keywords=15, min_words=2)
    keywords_counter.update(chunk_keywords)

for keyword in keywords_counter:
    keywords_2.append(keyword[0])
    print(keyword[0])
```

```
('natural', 'language', 'processing')
('machine', 'translation')
('external', 'links')
('computational', 'linguistics')
('information', 'retrieval')
('natural', 'language')
('artificial', 'intelligence')
('=', '=', 'references')
('computer', 'science')
('language', 'resources')
('machine', 'learning')
('information', 'extraction')
('speech', 'recognition')
('natural', 'languages')
('latent', 'semantic', 'analysis')
('natural', 'language', 'processing')
('text', 'mining')
('sentiment', 'analysis')
('word', 'sense', 'disambiguation')
('computational', 'linguistics')
('external', 'links')
('information', 'retrieval')
('natural', 'language')
('semantic', 'spaces')
('word', 'senses')
('=', '=', 'references')
('semantic', 'analysis')
('machine', 'learning')
('word', 'sense')
('sense', 'inventory')
```

[15]:

```
# This is a test cell, please ignore it!
```

6 Stop word based keyword extraction

One approach to extract keywords is by splitting the text at the stop words. Then we count these potential keywords and output the top n keywords. Make sure to only include words proper words. Here we define proper words as those words that match the regular expression `r'\b(\w{2,})\b'` (words that consist of at least 2 alphanumerical characters, including hyphens).

6.1 Stop word based keyword extraction A) [35 points]

Complete the function `keywords` in the class `StopWordKeywordExtractor`.

[14]:

```
%%time
from typing import List, Tuple
from collections import Counter
import re
import spacy

class StopWordKeywordExtractor:

    def __init__(self):
        # Set up SpaCy in a more efficient way by disabling what we do not need
        self.nlp = spacy.load('/srv/shares/NLP/spacy/en_core_web_sm',␣
 ↪disable=['ner', 'parser'])
        self.nlp.add_pipe('sentencizer')
        self.nlp.max_length = 1500000

    def is_proper_word(self, token: str) -> bool:
        '''
        Checks if the word is a proper word by our definition

        Arguments:
            token     -- The token as a string
        Return:
            is_proper -- True / False
        '''
        match = re.search(r'\b(\w{2,})\b', token)
        return bool(match) and token == match[0]

    def keywords(self, text: str, n_keywords: int, min_words: int) ->␣
 ↪List[Tuple[Tuple[str], int]]:
        '''
        Extract the top n most frequent keywords from the text.
        Keywords are sequences of adjectives and nouns that end in a noun

        Arguments:
            text       -- the raw text from which to extract keywords
            n_keywords -- the number of keywords to return
            min_words  -- the number of words a potential keyphrase has to␣
 ↪include
                          if this is set to 2, then only keyphrases consisting␣
 ↪of 2+ words are counted
        Returns:
            keywords   -- List of keywords and their count, sorted by the count
                          Example: [(('potato',), 12), (('potato',␣
 ↪'harvesting'), 9), ...]
        '''
        doc = self.nlp(text)
        stopwords = spacy.lang.en.stop_words.STOP_WORDS
        keywords = []
        candidate = []

        for token in doc:
            if token.text.lower() in stopwords:
                if candidate:
                    validated = [word for word in candidate if self.
 ↪is_proper_word(word)]
                    if len(validated) >= min_words:
                        keywords.append(tuple(validated))
                    candidate = []
            else:
                candidate.append(token.text.lower())

        if candidate:
            validated = [word for word in candidate if self.
 ↪is_proper_word(word)]
            if len(validated) >= min_words:
                keywords.append(tuple(validated))

        keyword_counter = Counter(keywords)
        most_common_keywords = keyword_counter.most_common(n_keywords)

        return most_common_keywords

with open('/srv/shares/NLP/datasets/wiki/wiki_nlp.txt', 'r') as corpus_file:
    text = corpus_file.read()

keywords = StopWordKeywordExtractor().keywords(text.lower(), n_keywords=15,␣
 ↪min_words=1)

'''
Expected output:
The keyword ('words',) appears 273 times.
The keyword ('text',) appears 263 times.
The keyword ('example',) appears 257 times.
The keyword ('word',) appears 201 times.
The keyword ('references',) appears 184 times.
The keyword ('natural', 'language', 'processing') appears 165 times.
...
'''
for keyword in keywords:
    print('The keyword {} appears {} times.'.format(*keyword))

# def process_file_in_chunks(file_path: str, chunk_size: int = 800000) -> str:
#     with open(file_path, 'r') as corpus_file:
#         while True:
#             chunk = corpus_file.read(chunk_size)
#             if not chunk:
#                 break
#             yield chunk

# extractor = StopWordKeywordExtractor()
# keywords_counter = Counter()

# for chunk in process_file_in_chunks('/srv/shares/NLP/datasets/wiki/wiki_nlp.txt'):
#     chunk_keywords = extractor.keywords(chunk.lower(), n_keywords=15, min_words=1)
#     keywords_counter.update(chunk_keywords)

# for keyword in keywords_counter:
#     print('The keyword {} appears {} times.'.format(*keyword))
```

```
The keyword ('words',) appears 219 times.
The keyword ('text',) appears 174 times.
The keyword ('example',) appears 163 times.
The keyword ('use',) appears 142 times.
The keyword ('set',) appears 139 times.
The keyword ('word',) appears 124 times.
The keyword ('number',) appears 102 times.
The keyword ('documents',) appears 86 times.
The keyword ('sentence',) appears 81 times.
The keyword ('language',) appears 80 times.
The keyword ('terms',) appears 76 times.
The keyword ('order',) appears 63 times.
The keyword ('applied',) appears 62 times.
The keyword ('meaning',) appears 62 times.
The keyword ('natural', 'language', 'processing') appears 61 times.
CPU times: user 18.8 s, sys: 2.85 s, total: 21.7 s
Wall time: 21.7 s
```

[17]:
```
# This is a test cell, please ignore it!
```

6.2 Stop word based keyword extraction B) [4 points]

Rerun the keyword extrator with a minimum word count of `min_words=2` and a keyword count of `n_keywords=15`.

Store this in the variable `keywords_2`. Print the result.

Make sure to convert the input text to lower case!

[15]:
```
keywords_2 = []

keywords = StopWordKeywordExtractor().keywords(text.lower(), n_keywords=15,␣
 ↪min_words=2)


for keyword in keywords:
    keywords_2.append(keyword[0])
    print(keyword[0])
```

```
('natural', 'language', 'processing')
('computational', 'linguistics')
('machine', 'translation')
('customer', 'inserts')
('natural', 'language')
('sentiment', 'analysis')
('computer', 'science')
('information', 'retrieval')
('machine', 'learning')
('artificial', 'intelligence')
('target', 'language')
('text', 'mining')
('language', 'resources')
('word', 'sense', 'disambiguation')
('speech', 'tagging')
```

[19]:
```
# This is a test cell, please ignore it!
```