Basic_text_preprocessing_with_SpaCy_handout_marthiensen

November 23, 2023

1 Using the SpaCy pipeline

This task is aiming to demonstrate the tokenization capabilites of SpaCy, as well as to serve as an introduction to the pipeline's capabilities combined with rule based matching.

Our goal will be to process the demonstration text, as well as to correct for some peculiarities, like special pronunciation marks, wide-spread abbreviations and foreign language insertions into our text.

It is mandatory, to stick to SpaCy based pipeline operations so as to make our analysis reproducible by running the pipeline on other texts presumably coming from the same corpus.

1.1 Our demonstration text

Original from Deutsche Sprache Wikipedia entry - with some modifications.

[1]:
```
text = '''Die deutsche Sprache bzw. Deutsch ([d_t ]; abgekürzt dt. oder dtsch.)␣
  ↪ist eine westgermanische Sprache.

And this is an English sentence inbetween.

Ihr Sprachraum umfasst Deutschland, Österreich, die Deutschschweiz,␣
  ↪Liechtenstein, Luxemburg, Ostbelgien, Südtirol, das Elsass und Lothringen␣
  ↪sowie Nordschleswig. Außerdem ist sie eine Minderheitensprache in einigen␣
  ↪europäischen und außereuropäischen Ländern, z. B. in Rumänien und Südafrika,␣
  ↪sowie Nationalsprache im afrikanischen Namibia.'''
```

1.2 Basic usage

After installing SpaCy, let us demonstrate it's basic usage by analysing our text.

[2]:
```
%%capture
!pip install tabulate >> /dev/null
!pip install spacy
```

[3]:
```
# Ok, we installed SpaCy, but do we have a model for German?
# Something has to be done here to get it!

# Running locally, it can be installed via the terminal as follows:
# python -m spacy download de_core_news_sm
# This could look different in Colab
```

[4]:

```
# If after the action above, the German model does not load, you may have to␣
  ↪restart the runtime.
# (in Colab it can be so)
```

[5]:

```
# Please do the appropriate imports for SpaCy and it's rule based Matcher class␣
  ↪and Language!
import spacy
from spacy.matcher import Matcher

# Please don't forget to instantiate the language model that we will use later␣
  ↪on for analysis
nlp = spacy.load('de_core_news_sm')
```

[6]:

```
# And please use the model to analyse the text from above!
doc = nlp(text)
```

1.2.1 Helper functions for nice printout

We just define some helper functions for nice printout. Nothing to do here, except to observe the ways one can iterate over a corpus or sentence, as well as the nice output of Tabulate.

[7]:

```
from tabulate import tabulate

def print_sentences(doc):
    for sentence in doc.sents:
        print(sentence,"\n")

def print_tokens_for_sentence(doc,sentence_num, stopwords=False):
    attribs=[]
    for token in list(doc.sents)[sentence_num]:
        if token.has_extension("is_lemma_stop"):
            if stopwords and token._.is_lemma_stop:
                pass
            else:
                attribs.append([token.text, token.lemma_, token.pos_])
        else:
            attribs.append([token.text, token.lemma_, token.pos_])
    print(tabulate(attribs))
```

[8]:

```
print_sentences(doc)
```

```
Die deutsche Sprache bzw. Deutsch ([d_t ];

abgekürzt dt.

oder dtsch.) ist eine westgermanische Sprache.
```

```
And this is an English sentence inbetween.



Ihr Sprachraum umfasst Deutschland, Österreich, die Deutschschweiz,
Liechtenstein, Luxemburg, Ostbelgien, Südtirol, das Elsass und Lothringen sowie
Nordschleswig.

Außerdem ist sie eine Minderheitensprache in einigen europäischen und
außereuropäischen Ländern, z. B. in Rumänien und Südafrika, sowie
Nationalsprache im afrikanischen Namibia.
```

[9]:

```
print_tokens_for_sentence(doc,-1)
```

```
-------------------  -------------------  -----
Außerdem             außerdem             ADV
ist                  sein                 AUX
sie                  sie                  PRON
eine                 ein                  DET
Minderheitensprache  Minderheitensprache  NOUN
in                   in                   ADP
einigen              einiger              DET
europäischen         europäisch           ADJ
und                  und                  CCONJ
außereuropäischen    außereuropäisch      ADJ
Ländern              Land                 NOUN
,                    --                   PUNCT
z.                   z.                   ADP
B.                   B.                   NOUN
in                   in                   ADP
Rumänien             Rumänien             PROPN
und                  und                  CCONJ
Südafrika            Südafrika            PROPN
,                    --                   PUNCT
sowie                sowie                CCONJ
Nationalsprache      Nationalsprache      NOUN
im                   in                   ADP
afrikanischen        afrikanisch          ADJ
Namibia              Namibia              PROPN
.                    --                   PUNCT
-------------------  -------------------  -----
```

1.3 Matching “zum Beispiel”

We are a bit frustrated, that the standard analysis pipeline does not know, that in German, “z. B.” is the abbreviation of “zum Beispiel” (like eg. is for “for example”), thus we would like to correct this.

Our approach is to extend the pipeline and do a matching, whereby we replace the `lemma` form of “z. B.” to the appropriate long form.

IMPORTANT design principle by SpaCy is, that one **always keeps the possibility to restore the original text**, so we are **NOT to modify `token.text`**. In the analysed form, we can do whatever we want.

It is typical to add layers to the pipeline which modify the analysis.

For our purposes, we will use rule based matching to achieve our goals.

A detailed description on rule based matching in SpaCy can be found here, or here

1.3.1 Build the matcher

With the help of rule based matching we create a matcher that reacts to the presence of “z. B.” exactly, then we use this matcher to define a pipeline step, that after matching, replaces the lemmas of the tokens “z.” and “B.” to their full written equivalent.

[10]:

```
zb_matcher = Matcher(nlp.vocab) # Please instantiate a matcher with the␣
 ↪appropriate parameters - think about all the words of the corpus...
# Please add an appropriate pattern to the matcher to match "z. B."
zb_matcher.add("zb_match", [[{"TEXT": "z."}, {"TEXT": "B."}]])



@spacy.Language.component("zb_replacer")
def zb_replacer(doc):
    matched_spans = []
    # Please use the matcher to get matches!
    matches = zb_matcher(doc)
    # Plsease iterate over the matches!
    for match_id, start, end in matches:
        span = doc[start:end] # get the span of text based on the matches␣
 ↪coordinates!
        matched_spans.append(span)
        print("ZB MATCH!!!")

    # Please iterate over matched spans
    for span in matched_spans:
        # And replace their lemmas to the appropriate ones!
        for token in span:
            if token.text == "z.":
                token.lemma_ = "zum"
            elif token.text == "B.":
                token.lemma_ = "Beispiel"
        # Please observe, that you don't have the ID of the desired lemmas,␣
↪just the their string form.
    return doc
```

1.3.2 Register it to the pipeline

After creating this processing step, we register it to be part of the pipeline and then run our analysis again.

[11]:
```
# Plase register the new zb_replacer to the pipeline!
# Think about, where to place it!
nlp.add_pipe("zb_replacer", last=True)
```

[11]:
```
<function __main__.zb_replacer(doc)>
```

1.3.3 Re-do the analysis and observe results

[12]:
```
doc=nlp(text)
```

```
ZB MATCH!!!
```

[13]:
```
print_tokens_for_sentence(doc,-1)
```

```
-------------------  -------------------  -----
Außerdem             außerdem             ADV
ist                  sein                 AUX
sie                  sie                  PRON
eine                 ein                  DET
Minderheitensprache  Minderheitensprache  NOUN
in                   in                   ADP
einigen              einiger              DET
europäischen         europäisch           ADJ
und                  und                  CCONJ
außereuropäischen    außereuropäisch      ADJ
Ländern              Land                 NOUN
,                    --                   PUNCT
z.                   zum                  ADP
B.                   Beispiel             NOUN
in                   in                   ADP
Rumänien             Rumänien             PROPN
und                  und                  CCONJ
Südafrika            Südafrika            PROPN
,                    --                   PUNCT
sowie                sowie                CCONJ
Nationalsprache      Nationalsprache      NOUN
im                   in                   ADP
afrikanischen        afrikanisch          ADJ
Namibia              Namibia              PROPN
```

```
.                    --                   PUNCT
-------------------  -------------------  -----
```

1.4 What are those ugly pronunciation signs doing there?

OK, so far so good. Let's observe, what is the problem with the first sentence!

[14]:
```
doc=nlp(text)
```

```
ZB MATCH!!!
```

[15]:
```
print_tokens_for_sentence(doc,0)
```

```
--------  -------  -----
Die       der      DET
deutsche  deutsch  ADJ
Sprache   Sprache  NOUN
bzw.      bzw.     CCONJ
Deutsch   Deutsch  NOUN
(         --       PUNCT
[         [        PROPN
d⁀t      d⁀t    ADV
]         ]        PROPN
;         --       PUNCT
--------  -------  -----
```

As we can see, poor pipeline can not really cope with the pronunciation markings of the phonetic alphabet, and thus thinks, that the signs are representing a foreign proper noun.

We would like to remedy this, and since we do expect further texts from the corpus to contain these inserted phonetics, we would like to match, merge and replace.

1.5 Building up matcher for PRONUNCIATION

To be more specific, we again first build up a matcher, that aims at the "square brackets" markings around the pronunciation. The task is to match everything between square brackets, or to be more specific: **everything that starts with an opening square bracket, and finishes with ";"**.

This matcher can then be used to:

1. Merge the resulting matching `span` into one token
2. Replace the token's lemma to "PRONUNCIATION"

For this to be achievable, we have to first register "PRONUNCIATION" as part of the vocabulary, moreover mark it as "stopword". (More on SpaCy's stopword handling here) See below.

[16]:
```
# Please instantiate and build the matcher as before with the appropriate pattern!
# Make it so, that the pattern will match ALL future pronunciations, not just the present one!
pron_matcher = Matcher(nlp.vocab)
```

```
pron_matcher.add("pron_match", [[{"ORTH": "["}, {}, {"ORTH": "]"}, {"ORTH": ";"}]])

# We set the properties for the new word "PRONUNCIATION"
lex = nlp.vocab['PRONUNCIATION']
lex.is_stop = True

@spacy.Language.component("pronunciation_replacer")
def pronunciation_replacer(doc):

    # Using the template above, please build a pronunciation replacer, that
    # 1. Gets the matches
    # 2. Iterates through them
    # 3. Collects them into a list
    # 4. initalizes - WITH context manager - a retokenizer
    # 5. goes through the spans
    # 6. merges them
    # 7. sets the lemma of the merged span to the PRONUNCIATION lemma we created before

    matched_spans = []
    # Please use the matcher to get matches!
    matches = pron_matcher(doc)
    # Please iterate over the matches!
    for match_id, start, end in matches:
        span = doc[start:end]  # get the span of text based on the matches coordinates!
        matched_spans.append(span)
        print("PRON MATCH!!!")

    # Collect matched spans
    with doc.retokenize() as retokenizer:
        for span in matched_spans:
            # Merge spans and set the lemma of the merged span to PRONUNCIATION
            retokenizer.merge(span, attrs={"LEMMA": "PRONUNCIATION"})

    return doc

nlp.add_pipe("pronunciation_replacer", after="zb_replacer")
```

```
[16]: <function __main__.pronunciation_replacer(doc)>
```

1.5.1 Observing result

[17]:

```
doc=nlp(text)
print_tokens_for_sentence(doc,0)
```

```
ZB MATCH!!!
PRON MATCH!!!
--------  -------------  -----
Die       der            DET
deutsche  deutsch        ADJ
Sprache   Sprache        NOUN
bzw.      bzw.           CCONJ
Deutsch   Deutsch        NOUN
(         --             PUNCT
[d_t ];  PRONUNCIATION  PROPN
--------  -------------  -----
```

In the future, we decide, we would not want to include the pronunciation tokens in our view. So we have to mark them as wtopwords.

1.5.2 Registering PRONUNCIATION as a stopword

Stopwords are typically those words, which do not contribute to the meaning of the sentence, are just there for syntactic reasons. There is a vague running list of these for languages. We will use and extend the German one in SpaCy.

[18]:

```
# import stop words from GERMAN language data
from spacy.lang.de.stop_words import STOP_WORDS

# Add PRONUNCIATION to stopwords
STOP_WORDS.add("PRONUNCIATION")
```

But since we will only be able to manipulate the lemmas of the pronunciation markings, we would have to let SpaCy know, that - in contrast to the default behavior, where stopwords are filtered on `text` level, we would like to have a new property for words, that is based on `lemma` level stopword filtering.

For these we will use extensions!

For more info please see here!

[19]:

```
from spacy.tokens import Token

# Please define a function (or lambda expression!) that checks if a Token, or␣
 ↪its lower case for,
# OR it's lemma string is contained it he stopword list above.
stop_words_getter = lambda token: str(token) in STOP_WORDS or token.lower_ in␣
 ↪STOP_WORDS or token.lemma_ in STOP_WORDS
```

```
# Set the above defined function as a extension for Token under the name␣
 ↪"is_lemma_stop" as a getter!
Token.set_extension("is_lemma_stop", getter=stop_words_getter)
```

[20]:

```
doc=nlp(text)
```

```
ZB MATCH!!!
PRON MATCH!!!
```

[21]:

```
print_tokens_for_sentence(doc,0, stopwords=True)

assert len(list(doc.sents)[0]) == 7
# Changed from 10 to 7, changes with specific model (_lg, _trf, _md, _sm)
```

```
--------  -------  -----
deutsche  deutsch  ADJ
Sprache   Sprache  NOUN
bzw.      bzw.     CCONJ
Deutsch   Deutsch  NOUN
(         --       PUNCT
--------  -------  -----
```

1.6 Language detection

We could also observe, that there is some English text inbetween our nice German sentences. We would like to detect foreign sentences and by later processing, ignore / skip them.

For this to be achievable, we need some language detection capabilities.

Luckily enough, we can make it part of our pipeline via this extension.

1.6.1 Standard installation

[22]:

```
%%capture
!pip install spacy-langdetect
```

[23]:

```
#Please import the language detector!
from spacy.language import Language
from spacy_langdetect import LanguageDetector
```

1.6.2 Adding language detection to our pipeline

[24]:

```
# Please register it to the pipeline as the final step of processing!
def get_lang_detector(nlp, name):
    return LanguageDetector()
Language.factory("language_detector", func=get_lang_detector)
nlp.add_pipe('language_detector', last=True)
```

[24]:

```
<spacy_langdetect.spacy_langdetect.LanguageDetector at 0x2c41b9a90>
```

1.6.3 Observing results

[25]:
```
doc = nlp(text)
```

```
ZB MATCH!!!
PRON MATCH!!!
```

[26]:
```
attribs = []
for sentence in doc.sents:
    attribs.append([list(sentence)[:5],"...", sentence._.language])
print(tabulate(attribs))

# Please observe how one accesses anextension!!
```

```
-----------------------------------------------  ---
-----------------------------------------------
[Die, deutsche, Sprache, bzw., Deutsch]          …  {'language': 'de',
'score': 0.9999976445750279}
[abgekürzt, dt, .]                               …  {'language': 'de',
'score': 0.9999974069611203}
[oder, dtsch, ., ), ist]                         …  {'language': 'de',
'score': 0.9999970913786898}
[And, this, is, an, English]                     …  {'language': 'en',
'score': 0.9999960445349885}
[Ihr, Sprachraum, umfasst, Deutschland, ,]       …  {'language': 'de',
'score': 0.9999979184291637}
[Außerdem, ist, sie, eine, Minderheitensprache]  …  {'language': 'de',
'score': 0.9999964866268759}
-----------------------------------------------  ---
-----------------------------------------------
```

2 Creating final generator for cleaned text

Typically for a later stage of NLP, we would like to have a generator like function, which allows us to iteratively access the corpus, albeit in it's cleaned and encoded form. Integer encoding (as well as one hot encoding) are quite typical representations of text.

In this spirit, we would like to implement a generator, that gives back an **array of lemmas OR lemma IDs for each sentence in the corpus, filtering out non-German sentences and punctuation / space marks**.

[27]:
```
# Please implement a generator function that yields the text of the corpus as␣
 ↪lists of sentences
# Based on the parameters either as a list of strings or a list of IDs
# It should filter out non-German sentences
# as well as topwords based on lemmas
# and punctuation and "space like" characters!

def sentence_generator(doc, ids=False):
    clean_ger_sent_lem = []
    # Split to individual sentences
    for sentence in doc.sents:
        # Filter out english, stopwords, punctuation, and "space like"␣
↪characters
        if sentence._.language["language"] == 'de':
            if ids:
                tokens_lemid = [token.lemma for token in sentence if not token.
↪is_stop and not token.is_punct and not token.is_space]
                clean_ger_sent_lem.append(tokens_lemid)
            else:
                tokens_lem = [token.lemma_ for token in sentence if not token.
↪is_stop and not token.is_punct and not token.is_space]
                clean_ger_sent_lem.append(tokens_lem)

    return clean_ger_sent_lem
```

[28]:

```
for i in sentence_generator(doc):
    print(i,"\n")

for i in sentence_generator(doc, ids=True):
    print(i,"\n")
```

```
['deutsch', 'Sprache', 'bzw.', 'Deutsch', 'PRONUNCIATION']

['abkürzen', 'dt']

['dtsch', 'westgermanisch', 'Sprache']

['Sprachraum', 'umfassen', 'Deutschland', 'Österreich', 'Deutschschweiz',
'Liechtenstein', 'Luxemburg', 'Ostbelgie', 'Südtirol', 'Elsass', 'Lothringen',
'Nordschleswig']

['Minderheitensprache', 'europäisch', 'außereuropäisch', 'Land', 'zum',
'Beispiel', 'Rumänien', 'Südafrika', 'Nationalsprache', 'afrikanisch',
'Namibia']

[5968319817064592459, 8431935777423264011, 3072869516764223635,
13347145995516113707, 2111469962564940766]

[12068858602874567954, 5135506797272647618]

[2552743035069842888, 1774304420854013574, 8431935777423264011]

[11854469037278879099, 6826961035611069329, 3491614202785599281,
16047064563126251420, 3469156011154928224, 10833980334450146958,
15216956676957942053, 5534291137827076893, 14425170055224073740,
14854674721094831692, 5682654018506929560, 10694615845175474381]

[13853446524293058697, 512110525822973470, 15751849195492229329,
731233208058718707, 11601248322003946775, 176351906757609250,
16018282812866072734, 14398131728093720111, 13884865873598079458,
9226656959411645728, 2911802427415368037]
```