curran_thomas_assignment1-2

August 29, 2021

1 Natural Language Processing

Tom Curran
August 30, 2021
Homework 1 - 2

Assignment:

There are roughly 30k entries in the standard BERT vocabulary download. In this HW, you should explore BERT's vocabulary to get a feel for what's really in there. Structure your exploration around the notion of how many "words" are really in its vocabulary. Follow along the path that's started in the notebook presented in lecture. Specifically, assume that the special reserved entries ([…]), numbers, subwords, and single characters are not words. From there you should explore the remaining full "words" to come up with a coherent rationale for your final count.

Submit a short report that describes your process for coming up with your estimate. You don't need to use unix commands to explore the vocabulary. Feel free to use Python, some other language, or an editor that supports regular expressions.

In order to determine BERT's count of actual words we will use python (v3) and the natural language processing libraries `SpaCy` and `NLTK`. We will also use the python `re` module for regex operations.

[165]:
```
import re
from pprint import pprint
import pandas as pd
import nltk
from nltk.stem.porter import *
import spacy
nlp = spacy.load('en_core_web_sm')
```

For text pre-processing, we will open each item in BERT-vocab.txt as its own line. This allows for easier filtering of things we should not consider "words". We will also automatically strip any new line characters from the outset.

[166]:
```
with open('BERT-vocab.txt') as file:
    vocab = file.readlines()

# remove the newline characters for all strings before limiting list down
```

```
bert = [item.strip() for item in vocab]
print("Intiial Number of Items in Bert-vocab: {}".format(len(bert)))
```

```
Intiial Number of Items in Bert-vocab: 30522
```

[167]:
```
bert[:10]
```

[167]:
```
['[PAD]',
 '[unused0]',
 '[unused1]',
 '[unused2]',
 '[unused3]',
 '[unused4]',
 '[unused5]',
 '[unused6]',
 '[unused7]',
 '[unused8]']
```

We will use the `non_words` array to collect any items in the BERT vocabulary that shouldn't count as words. For this first pass we will remove any item in the vocabulary that starts with '['. Words that do not contain this character will be collected and appended to the array `words1`

[168]:
```
non_words = []
words1 = []
```

[169]:
```
for item in bert:
    if re.findall('^\[', item):
        non_words.append(item)
    else:
        words1.append(item)

print("Number of non-words found: {}".format(len(non_words)))
print("Number of items left in BERT-vocab: {}".format(len(words)))
```

```
Number of non-words found: 1000
Number of items left in BERT-vocab: 21725
```

So far we have removed 1,000 entries from the BERT-vocab that do not count as a word

Next, we remove any symbols, punctunctuation and single character words since they defintionally do not count as words. As we can can see from the resuts of the first round of pre-processing, things like the exclamation point, question mark and other grammatical devices will qualify for removal in this next round of preprocessing

[170]:
```
words1[:10]
```

[170]:
```
['!', '"', '#', '$', '%', '&', "'", '(', ')', '*']
```

[171]:
```
words2 = []
```

```
for item in words1:
    if re.findall('^.$', item):
        non_words.append(item)
    else:
        words2.append(item)
```

[172]:
```
words2[0:10]
```

[172]: ['the', 'of', 'and', 'in', 'to', 'was', 'he', 'is', 'as', 'for']

[173]:
```
len(non_words)
```

[173]: 1996

[174]:
```
len(words2)
```

[174]: 28526

After the second round of pre-processing, we have removed an approximately another ~1000 items from the BERT-vocab file. This leaves approximately ~28500 words left in BERT's vocabulary. We can see that the BERT-vocab file contains suffixes indicated by the '##' symbol. We will remove those from the vocabulary list since they do not count as full words

[175]:
```
sorted(words2)[:10]
```

[175]: ['##!', '##"', '###', '##$', '##%', '##&', "##'", '##(', '##)', '##*']

[176]:
```
words3 = []

for item in words2:
    if re.findall('^##', item):
        non_words.append(item)
    else:
        words3.append(item)
```

[177]:
```
words3[0:10]
```

[177]: ['the', 'of', 'and', 'in', 'to', 'was', 'he', 'is', 'as', 'for']

You can see below that we have succesfully removed items that start with '##'

[178]:
```
non_words[-10:]
```

[178]: ['## ', '## ', '## ', '## ', '## ', '## ', '## ', '## ', '## ', '## ']

[179]:
```
len(non_words)
```

[179]: 7824

[180]:
```python
len(words3)
```

[180]: 22698

We have removed approximately another ~6000 words from the BERT-vocab file to get closer to only containing 'words'

[181]:
```python
sorted(words3)[0:10]
```

[181]: ['…', '00', '000', '001', '00pm', '01', '02', '03', '04', '05']

Now, we can see that BERT-vocab is also counting items that are numbers or contain digits. We should not count these as words and should therefore be removed from the vocabulary list

[182]:
```python
words4 = []

for item in words3:
    if re.findall('\d', item):
        non_words.append(item)
    else:
        words4.append(item)
```

[183]:
```python
sorted(words4)[0:10]
```

[183]:
```
['…',
 'aa',
 'aaa',
 'aachen',
 'aarhus',
 'aaron',
 'ab',
 'aba',
 'aback',
 'abandon']
```

[184]:
```python
len(words4)
```

[184]: 21731

[185]:
```python
len(non_words)
```

[185]: 8791

We have removed approximately another ~1000 entries from the BERT-vocab. Now, in our last round of pre-processing, we will make that the remaining items contain only letters a through z and are greater than length 1. this will finalize our list of items to count the actual vocabulary of BERT-vocab

[186]:
```
words5 = []

for item in words4:
    if re.findall('^[a-zA-Z]+$', item) and len(item)>1:
        words5.append(item)
    else:
        non_words.append(item)
```

[187]:
```
sorted(words5)[0:10]
```

[187]:
```
['aa',
 'aaa',
 'aachen',
 'aarhus',
 'aaron',
 'ab',
 'aba',
 'aback',
 'abandon',
 'abandoned']
```

[188]:
```
sorted(words5)[-10:]
```

[188]:
```
['zones',
 'zoning',
 'zoo',
 'zoological',
 'zoology',
 'zoom',
 'zu',
 'zulu',
 'zur',
 'zurich']
```

[189]:
```
len(words5)
```

[189]:
```
21719
```

[190]:
```
len(non_words)
```

[190]:
```
8803
```

We can see from the latest entries int the `non_words` array that there are some items in the BERT-vocab that contains both characters and non-characters. We have succesfully filtered BERT-vocab to only contain "words" that are characters with length (i.e. number of characters) greater than 1/

[191]:
```
non_words[-10:]
```

[191]: ['°c', 'm²', '°f', 'm³', 'łodz', '¹/', 'co', 'h o', 'wrocław', 'stanisław']

1.0.1 Counting Number of Words:

Now that we have preprocessed the BERT-vocab to only include "words" that are of length greater than 1 and only contain letters, we can count the actualy number of words that are contained within BERT-vocab.

We will use the `SpaCy` and `NLTK` python packages to accomplish this.

First, we will use `SpaCy` to count only the "lemmas" for each of the words in BERT-vocab. To accomplish this, we must first convert the list of words that we refined from the original BERT-vocab list into a tokenized `SpaCy` object

[87]:
```
word_corpus = nlp(' '.join(words5))
```

[193]:
```
print(type(word_corpus))
```

```
<class 'spacy.tokens.doc.Doc'>
```

Converting to a `SpaCy` tokenized object allows doesn't alter the actual contents of the list. As you can see below the length of the spacy corpus is the same from the pre-processed BERT-vocab list

[192]:
```
len(word_corpus)
```

[192]: 21725

Using `SpaCy`'s `.lemma_` method for each token in the corpus, we can take the lemma of each word taken from the pre-processed BERT-vocab file. A lemma, as defined by Stanford University's Natural Language Processing Group is, "…refers to 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…". 1

Finding the lemmas will essentialy allow use to count the number of words without worrying about repeated words that only look different due to tenses.

for example, if we look for words that start with "look", we see that there are several versions from the pre-processed BERT-vocab

[195]:
```
for item in words5:
    if re.findall('^look', item):
        print(item)
```

```
looked
look
looking
looks
lookout
```

As we can see that "looked", "look", "looking" and "looks" all have a similar root word of "look". Using `SpaCy`'s `lemma_` method we can normalize the text to count the number of "words" that are included in the BERT-vocab

[196]:
```
# for each word, append its original lemma to the lemmas array
lemmas = []

# keep the original word in the words array
words = []
```

[197]:
```
for word in word_corpus:
    lemmas.append(word.lemma_)
    words.append(word.text)
```

An alternative to lemmatization of words, is *stemming*. *Stemming* is a more niave version of lemmatization of tokens that "…consists of chopping off word-final affixes". Here, we use the Porter Stemming algorithm. Though more "crude" than the lemmatization approach, we want to explore the stemming methods as a means to truly understand how many "words" are contained in BERT-vocab.txt. Like lemmas, we use stemming a means to circumvent double counting pre-processed words that are subject to different morphologies.

We are using the Porter Stemmer from the python package `nltk`.

[101]:
```
stemmer = PorterStemmer()
```

[120]:
```
porter_stems = [stemmer.stem(word) for word in words]
```

[121]:
```
porter_stems[0:10]
```

[121]:
```
['the', 'of', 'and', 'in', 'to', 'wa', 'he', 'is', 'as', 'for']
```

Now that we have gone through the preprocessing of BERT-vocab and passed the cleaned data through the lemma and stem methods, we create a pandas dataframe to hold the revelant information.

[129]:
```
df = pd.DataFrame({
    'word':words,
    'lemma':lemmas,
    'porter_stem':porter_stems
})
```

[130]:
```
df
```

[130]:

	word	lemma	porter_stem
0	the	the	the
1	of	of	of
2	and	and	and
3	in	in	in
4	to	to	to
…	…	…	…
21720	nitrate	nitrate	nitrat
21721	salamanca	salamanca	salamanca
21722	scandals	scandal	scandal

```
21723       thyroid      thyroid     thyroid
21724  necessitated  necessitate    necessit

[21725 rows x 3 columns]
```

[131]:
```
unique_lemmas = df.lemma.value_counts()
pprint(unique_lemmas)
```

```
be             7
close          6
model          6
get            6
travel         6
              ..
lineage        1
oct            1
cody           1
hansen         1
necessitate    1
Name: lemma, Length: 16395, dtype: int64
```

Here, we can see that the most common lemma is `be`, which intuitively makes sense. If you look at the words attached to the lemma you can clearly see how we have reduced the number of words that BERT-vocab knows because the words are simply a morphed version of be

[200]:
```
df[df.lemma == 'be'].reset_index(drop=True)
```

[200]:

	word	lemma	porter_stem
0	was	be	wa
1	is	be	is
2	were	be	were
3	be	be	be
4	are	be	are
5	been	be	been
6	being	be	be

[203]:
```
print("Number of words that BERT-vocab knows based on lemmas: {}".
 ↪format(len(unique_lemmas.keys())))
```

```
Number of words that BERT-vocab knows based on lemmas: 16395
```

Based on the Porter Stemming Algorithm, we see that the most used stem in the BERT-vocab.txt file is `gener`

[134]:
```
unique_stems = df.porter_stem.value_counts()
print(unique_stems)
```

```
gener       14
organ       13
oper        11
```

```
commun       11
travel       10
             ..
centimetr     1
gm            1
baghdad       1
batsman       1
necessit      1
Name: porter_stem, Length: 14202, dtype: int64
```

[204]:
```python
print("Number of words that BERT-vocab knows based on Porter Stems: {}".
 ↪format(len(unique_stems.keys())))
```

```
Number of words that BERT-vocab knows based on Porter Stems: 14202
```

1.0.2 Conclusion

In the raw BERT-vocab.txt file, there are 30,522 tokens. Further exploration revlead that there were tokens in that file that did not match our definition of a "word". In this case we are focusing only on words in english and must be contain only characters (A through Z) and has more than 1 character.

From the original 30,522 tokens, we preprocessed the tokens down to 21,725. Of the preprocessed text, there were instances where the tokens were morphologies of the same word (i.e. look, looked, looking, etc.). As such, to determines how many words BERT-vocab actually has we looked at the lemmas and (porter) stems of the words in the pre-processed BERT-vocab. Since the stem does not always constitute a "usable" word, I would use the number of lemmas as a more reliable metric for number of actual words known by BERT-vocab.

Therefore, we can estimate that BERT-vocab knows approximately 16395 words