p_model_DR

May 29, 2023

[1]:
```
import nltk
from nltk.probability import FreqDist
from nltk.tokenize import word_tokenize
from bs4 import BeautifulSoup
from urllib import request
from collections import Counter
# nltk.download('punkt')

stopwords = nltk.corpus.stopwords.words('english')
```

[2]:
```
url = 'https://www.gutenberg.org/files/8300/8300-h/8300-h.htm'
# This is the html of the entire Douay-Rheims Bible
html = request.urlopen(url).read().decode('utf8')
# Here we are turning it into a string that Beautiful Soup will be able to work␣
 ↪with
```

[3]:
```
soup = BeautifulSoup(html, 'html.parser')
# Beautiful Soup will parse the html and make it where we can pull out just the␣
 ↪bits we need
```

[4]:
```
p_list = [text for text in soup.stripped_strings]
# Using a list comprehension to make a list of each unique <p>
# After several other methods, this seems to be the easiest way to single out␣
 ↪the actual verses from the commentary, titles, etc..



# for elm in p_list[228:250]:
#     print(elm)
#     print('\n')
# print(p_list[-57:])

# This whole section above was a guess-and-check to find where the Bible begins␣
 ↪and ends.
```

[5]:
```
dr_p_cln = []
# I will need a list to store items from the p_list above "paragraph"_list or␣
 ↪<p>_list, I'm only storing the ones that have a numerical value as their␣
 ↪first item
```

```
num_list = []
for i in range(0,100):
    num_list.append(str(i))

# Here I am constructing a list of more than all possible numbers we might see␣
 ↪in a chapter:verse
# Note that we are transforming the int objects into str objects, this is so␣
 ↪they'll recognize two numbers as the same objects

for verse in p_list[228:-57]:
    if verse[0] in num_list:
        dr_p_cln.append(verse)

# I am iterating through the the portion of p_list that contains actual␣
 ↪scripture
# I am selecting only <p>s that have a number as their initial value
# I am adding them to the list created above, thusly we have a list of all and␣
 ↪only the verses of the Douay-Rheims. The hard part is now done.

# print(dr_p_cln[:15])
# Checking work
```

[6]:
```
dr_2 = []

# Here is a list where we will put the verses as objects after removing␣
 ↪numbers, new line markers, and punctuation marks

repl_me = ['\n','\r',',','.',';',':','"','-','--','!','?','"','"','-
',"'",'(',')','[',']']

# A list of punctuation marks and new line markers that we will be removing

for i in range(0,100):
    repl_me.append(str(i))

# We are adding strings of the integers in the same method as before to the␣
 ↪list 'repl_me'

for verse in dr_p_cln:
    str_ver = verse
    for k in range(0, len(repl_me)):
        str_ver = str_ver.replace(repl_me[k],'')
    dr_2.append(str_ver)

# We are iterating over every object in dr_p_cln
# First we set each verse as a unique new object called 'str_ver'
# Then we iterate over the entire list of repl_me (which is 1-1 with␣
↪repl_me_dict, ie, it has the same number of objects)
# We update str_ver after replcing every object in the string that is in␣
↪repl_me [note the .replace(repl_me[k] selects for each object eventually]␣
↪replacing it with nothing: ''
# Then we append this updated str_ver to our created list dr_2


print(dr_2[:5])
# Checking our work!
```

```
[' In the beginning God created heaven and earth', ' And the earth was void and
empty and darkness was upon the             face of the deep and the spirit of
God moved over the waters', ' And God said Be light made And light was made', '
And God saw the light that it was good and he divided the            light from
the darkness', ' And he called the light Day and the darkness Night and there
was evening and morning one day']
```

[7]:
```
dr_t = []

# Here is a list for after we tokenize each word in dr_2 and turn it into a␣
↪list thtat is still ordered but can be treated as a 'bag of words' (see for␣
↪more info on this term: https://en.wikipedia.org/wiki/Bag-of-words_model)

for verse in dr_2:
    tokens_temp = word_tokenize(verse)
    for i in tokens_temp:
        dr_t.append(i)

# We iterate through every object in dr_2, which has each verse cleaned of␣
↪punctuation and numbers
# Using word_tokenize we tokenize each string in the list into individual word␣
↪objects and store them in the list 'tokens_temp'
# We then iterate over tokens_temp, appending each object to dr_t, the list we␣
↪created above. W
# We do this to avoid creating a list of lists which would be harder to process

print(dr_t[0:50])
# Checking our work!
```

```
['In', 'the', 'beginning', 'God', 'created', 'heaven', 'and', 'earth', 'And',
'the', 'earth', 'was', 'void', 'and', 'empty', 'and', 'darkness', 'was', 'upon',
'the', 'face', 'of', 'the', 'deep', 'and', 'the', 'spirit', 'of', 'God',
'moved', 'over', 'the', 'waters', 'And', 'God', 'said', 'Be', 'light', 'made',
'And', 'light', 'was', 'made', 'And', 'God', 'saw', 'the', 'light', 'that',
'it']
```

[8]:
```
dr_t2 = [w.lower() for w in dr_t]
# We make every word lowercase to make it easier for python to count instances

print(dr_t2[:5])
# Checking our work!
```

```
['in', 'the', 'beginning', 'god', 'created']
```

[9]:
```
fdist_0 = FreqDist(dr_t2)

# Here we are using FreqDist from nltk.probability to find the most common␣
 ↪words in our list

fdist_0.most_common(5)
# Checking our work!
# Notice the top 5 are all stopwords! Let's see what happens when we remove␣
 ↪them:
```

[9]:
```
[('the', 70874), ('and', 60171), ('of', 38666), ('to', 23703), ('in', 15502)]
```

[10]:
```
dr_t3 = [w for w in dr_t2 if w not in stopwords]

# We create a new list based on dr_t2 without the stopwords

fdist_1 = FreqDist(dr_t3)

# Using FreqDist from nltk.probability to find the most common words in our list

fdist_1.most_common(5)
# Checking our work!
# Notice these words are much more informative about the contents of the Bible
```

[10]:
```
[('shall', 11269),
 ('lord', 8221),
 ('thou', 6127),
 ('thy', 6052),
 ('god', 4945)]
```

[11]:
```
my_trigrams = nltk.trigrams(dr_t2)
# print(type(my_trigrams))

# This experiment is interested in trigrams in the Bible. We will look at both␣
 ↪trigrams with and without stop words
# Here we are using nltk.trigrams to analyze our list (still in order,␣
 ↪remember) for groups of 3 words in a row
# this object is a generator though, and to use it we'd like to turn it into a␣
 ↪list:
```

```
trigram_list = []
for i in my_trigrams:
    trigram_list.append(i)

# Here we create a list for the trigrams to be stored in
# Then we iterate over the generator and append each output into the new list

print(trigram_list[:5])
# Checking our work!
```

```
[('in', 'the', 'beginning'), ('the', 'beginning', 'god'), ('beginning', 'god',
'created'), ('god', 'created', 'heaven'), ('created', 'heaven', 'and')]
```

[12]:
```
a = dict(Counter(trigram_list))

# Here we are using Counter from collections in the creation of a dictionary
# For the keys we will have each unique trigram, and for the values we will␣
↪have a count on how many times that unique trigram occurred in 'trigram_list'
```

[13]:
```
b = {v:k for k,v in a.items()}

# Here we are flipping the keys and values to where we have the count of␣
↪instances of each unique trigram as our key, and the trigram as the value
# This will make sorting the dictionary and getting the top counts easier
```

[14]:
```
items_sorted = sorted(b.items(), reverse=True)

# Sorting the reversed dictionary, b.
# Reversing the order of values such as it will be descending in value for an␣
↪easier evaluation

print(items_sorted[0:50])

# The top 50 trigrams in the Douay-Rheims accompanied with their counts of how␣
↪many times they occurred
```

```
[(1890, ('of', 'the', 'lord')), (1486, ('the', 'son', 'of')), (1235, ('the',
'children', 'of')), (930, ('and', 'i', 'will')), (914, ('the', 'lord', 'and')),
(871, ('saith', 'the', 'lord')), (806, ('out', 'of', 'the')), (801, ('the',
'house', 'of')), (654, ('the', 'land', 'of')), (636, ('the', 'sons', 'of')),
(635, ('and', 'the', 'lord')), (633, ('to', 'the', 'lord')), (624, ('children',
'of', 'israel')), (597, ('and', 'all', 'the')), (548, ('said', 'to', 'him')),
(514, ('the', 'king', 'of')), (508, ('and', 'thou', 'shalt')), (505, ('and',
'they', 'shall')), (489, ('and', 'he', 'said')), (458, ('the', 'midst', 'of')),
(456, ('according', 'to', 'the')), (446, ('thus', 'saith', 'the')), (435,
('the', 'god', 'of')), (433, ('he', 'said', 'to')), (388, ('of', 'israel',
'and')), (382, ('and', 'he', 'shall')), (380, ('the', 'word', 'of')), (376,
('said', 'to', 'them')), (375, ('in', 'the', 'midst')), (370, ('of', 'the',
'earth')), (359, ('and', 'in', 'the')), (355, ('the', 'hand', 'of')), (347,
('and', 'when', 'he')), (346, ('the', 'lord', 'god')), (338, ('the', 'lord',
'hath')), (325, ('of', 'the', 'children')), (323, ('and', 'of', 'the')), (319,
('of', 'the', 'land')), (314, ('in', 'the', 'land')), (304, ('the', 'name',
'of')), (294, ('word', 'of', 'the')), (290, ('of', 'the', 'house')), (287,
('all', 'the', 'people')), (286, ('the', 'lord', 'thy')), (281, ('the', 'lord',
'of')), (280, ('the', 'earth', 'and')), (276, ('lord', 'thy', 'god')), (274,
('the', 'lord', 'the')), (273, ('house', 'of', 'the')), (272, ('of', 'god',
'and'))]
```

[15]:
```
a['be','not','afraid']

# I was personally curious how often this one occurred.
```

[15]:
```
17
```

[ ]:
```
# Below we have the same exact style of analysis, but utilizing the list with␣
↪the stop words removed, ie, dr_t3. This analysis proved less fruitful (as␣
↪you can see from the final output).
```

[16]:
```
my_trigrams_2 = nltk.trigrams(dr_t3)

# This experiment is interested in trigrams in the Bible. We will look at both␣
↪trigrams with and without stop words
# Here we are using nltk.trigrams to analyze our list (still in order,␣
↪remember) for groups of 3 words in a row
# this object is a generator though, and to use it we'd like to turn it into a␣
↪list:

trigram_list_2 = []
for i in my_trigrams_2:
    trigram_list_2.append(i)

# Here we create a list for the trigrams to be stored in
# Then we iterate over the generator and append each output into the new list

print(trigram_list_2[:5])
# Checking our work!
```

```
[('beginning', 'god', 'created'), ('god', 'created', 'heaven'), ('created',
'heaven', 'earth'), ('heaven', 'earth', 'earth'), ('earth', 'earth', 'void')]
```

[17]:
```
a_2 = dict(Counter(trigram_list_2))

# Here we are using Counter from collections in the creation of a dictionary
# For the keys we will have each unique trigram, and for the values we will␣
↪have a count on how many times that unique trigram occurred in␣
↪'trigram_list_2'
```

```
[18]: b_2 = {v:k for k,v in a_2.items()}

      # Here we are flipping the keys and values to where we have the count of␣
       ↪instances of each unique trigram as our key, and the trigram as the value
      # This will make sorting the dictionary and getting the top counts easier
```

```
[19]: val_sorted_2 = sorted(b_2.items(), reverse=True)

      # Sorting the reversed dictionary, b_2.
      # Reversing the order of values such as it will be descending in value for an␣
       ↪easier evaluation

      print(val_sorted_2[0:50])

      # The top 50 trigrams in the Douay-Rheims accompanied with their counts of how␣
       ↪many times they occurred, with the stop words removed
      # Personally I find this less useful than when the stop words were not removed.␣
       ↪I'm glad we did both!
```

```
[(433, ('thus', 'saith', 'lord')), (284, ('saith', 'lord', 'god')), (277,
('lord', 'thy', 'god')), (136, ('saith', 'lord', 'hosts')), (126, ('lord',
'god', 'israel')), (120, ('thee', 'thou', 'shalt')), (118, ('word', 'lord',
'came')), (91, ('lord', 'spoke', 'moses')), (89, ('thou', 'shalt', 'say')), (87,
('lord', 'jesus', 'christ')), (80, ('shall', 'know', 'lord')), (74, ('thou',
'shalt', 'make')), (69, ('thou', 'shalt', 'take')), (68, ('shall', 'come',
'pass')), (67, ('thou', 'hast', 'made')), (64, ('therefore', 'thus', 'saith')),
(63, ('lord', 'came', 'saying')), (60, ('thee', 'thou', 'hast')), (56, ('let',
'us', 'go')), (55, ('lord', 'said', 'moses')), (52, ('say', 'thus', 'saith')),
(51, ('shall', 'come', 'upon')), (50, ('nabuchodonosor', 'king', 'babylon')),
(49, ('thou', 'hast', 'done')), (47, ('mercy', 'endureth', 'ever')), (46,
('saying', 'thus', 'saith')), (45, ('saying', 'son', 'man')), (44, ('hosts',
'god', 'israel')), (41, ('neither', 'shalt', 'thou')), (40, ('lord', 'god',
'hosts')), (39, ('came', 'saying', 'son')), (38, ('son', 'reigned', 'stead')),
(37, ('old', 'began', 'reign')), (36, ('every', 'one', 'shall')), (35, ('lord',
'god', 'behold')), (34, ('book', 'words', 'days')), (33, ('words', 'days',
'kings')), (32, ('holy', 'one', 'israel')), (31, ('praise', 'exalt', 'ever')),
(30, ('saith', 'lord', 'shall')), (29, ('lord', 'praise', 'exalt')), (28,
('thy', 'people', 'israel')), (27, ('bless', 'lord', 'praise')), (26, ('jesus',
'answering', 'said')), (25, ('amen', 'amen', 'say')), (24, ('slept', 'fathers',
'buried')), (23, ('sons', 'brethren', 'twelve')), (22, ('thou', 'son', 'man')),
(21, ('let', 'every', 'man')), (20, ('father', 'lord', 'jesus'))]
```