CSCI-8450

2018 Spring

Yuchen Wang

<http://www.nltk.org/book/ch05.html>

1. Write programs to process the Brown Corpus and find answers to the following questions:
   1. Which nouns are more common in their plural form, rather than their singular form (Use the parts of speech tags in the corpus to identify plural versus singular nouns and use nltk.WordNetLemmatizer() to get the singular form of a noun from its plural form). List the five most frequent nouns that feature this property.

Code:

**def** findtags(tag\_prefix, tagged\_text):  
 cfd = nltk.ConditionalFreqDist((tag, word.lower()) **for** (word, tag) **in** tagged\_text **if** tag.startswith(tag\_prefix))  
 **return** dict((tag, cfd[tag].most\_common(5)) **for** tag **in** cfd.conditions())

brown\_words = brown.words()  
brown\_tags = brown.tagged\_words()  
tagdict = findtags(**'NNS'**, nltk.corpus.brown.tagged\_words()) *# a dictionary with a list of words have 'NNS' tag*total\_list = []  
**for** tag **in** sorted(tagdict):  
 total\_list.append(tagdict[tag])  
print(**"total\_list: "**, total\_list)  
  
total\_list\_w = list(chain.from\_iterable(total\_list))  
print(**"total\_list\_w:"**, total\_list\_w)  
total\_list\_sorted = sorted(total\_list\_w, key= operator.itemgetter(1), reverse = **True**)  
print(**"total\_list\_sorted"** , total\_list\_sorted)  
  
lem = nltk.WordNetLemmatizer()  
words = []  
first\_element = [x **for** x,\_ **in** total\_list\_sorted]  
  
print(**"first element"**, first\_element)  
cf = nltk.FreqDist(brown\_words)  
**for** i **in** first\_element:  
 singular = lem.lemmatize(i)  
 freq\_sing = cf[singular]  
 freq\_plur = cf[i]  
 **if** freq\_plur > freq\_sing:  
 words.append(i)  
print(**"words:"**, words[:5])

**Result:**

Part a

words: ['years', 'eyes', 'instructions', 'things', 'friends']

* 1. List the 5 most frequent tags in order of decreasing frequency. What do the tags represent?

Code:

print(**"Part b"**)  
tags = [b[1] **for** (a, b) **in** nltk.bigrams(brown\_tags)]  
fd = nltk.FreqDist(tags)  
print(fd.most\_common(5))  
print(**"The tags represent the decrease in frequency."**)

Result:

Part b

[('NN', 152470), ('IN', 120557), ('AT', 97958), ('JJ', 64028), ('.', 60638)]

The tags represent the decrease in frequency.

* 1. Which three tags precede nouns tagged with the 'NN' tag most commonly? What do these three tags represent? Report your findings separately for the following categories of Brown corpus: *humor, romance, government*.

Code:

print(**"Part c"**)  
categories = [**'humor'**, **'romance'**, **'government'**]  
  
**for** category **in** categories:  
  
 category\_tags = brown.tagged\_words(categories=category)  
 tagList = [a[1] **for** (a, b) **in** nltk.bigrams(category\_tags) **if** b[1].startswith(**'N'**) **and** b[1] != **'N'**]  
 fd = nltk.FreqDist(tagList).most\_common()  
 first\_element = [x **for** x, \_ **in** list(fd)]  
 print (category, **', '**.join(first\_element[:3]))

Result:

Part c

humor AT, JJ, IN

romance AT, JJ, IN

government AT, JJ, IN

1. In the “Combining Taggers” Subsection of Section 5.5 of the textbook, an example of a backoff tagger is provided. Extend that example by defining a TrigramTagger called t3 which backs off to t2. Train this tagger on all of the sentences from the Brown corpus with the category *news.* Then
2. evaluate your tagger using “evaluate” function on all of the sentences from the Brown corpus with the category *lore.* Report the number.How does this number compare to when this tagger is evaluated on all of the sentences from the Brown corpus with the category *news*.

Code:

news\_tagged\_sents = brown.tagged\_sents(categories=**'news'**)  
t0 = nltk.DefaultTagger(**'NN'**)  
t1 = nltk.UnigramTagger(news\_tagged\_sents, backoff=t0)  
t2 = nltk.BigramTagger(news\_tagged\_sents, backoff=t1)  
t3 = nltk.TrigramTagger(news\_tagged\_sents, backoff=t2)  
news\_test\_sents = t3.evaluate(news\_tagged\_sents)  
print (news\_test\_sents)

print(**"Part a"**)  
lore\_tagged\_sents = brown.tagged\_sents(categories=**'lore'**)  
  
lore\_tagger = t3.evaluate(lore\_tagged\_sents)

print(**"Compare DefaultTagger of lore and news:"**,lore\_tagger, news\_test\_sents)

Result:

Part a

Compare DefaultTagger of lore and news: 0.8427274952628764 0.9826759750979573

1. Provide the output of your tagger on the 200th sentence of the *lore* category of the Brown Corpus (note how brown.sents(categories='lore')[199] produces the 200th sentence). Would you tag this sentence in the same manner?

Code:

print(**"Part b"**)  
lore\_size = 199 *# 200th sentence*lore\_train\_sents = lore\_tagged\_sents[:lore\_size]  
lore\_test\_sents = lore\_tagged\_sents[lore\_size:]  
  
unigram\_tagger = nltk.UnigramTagger(lore\_tagged\_sents)  
unigram\_val = unigram\_tagger.evaluate(lore\_tagged\_sents)  
  
bigram\_tagger = nltk.BigramTagger(lore\_train\_sents)  
bigram\_val = bigram\_tagger.evaluate(lore\_test\_sents)  
  
trigram\_tagger = nltk.BigramTagger(lore\_train\_sents)  
trigram\_val = trigram\_tagger.evaluate(lore\_test\_sents)  
print(t3.tag(brown.sents(categories=**'lore'**)[199]))  
*# print(brown.sents(categories='lore')[199])*print(**"Unigram"**, unigram\_val, **'vs.Bigram'**, bigram\_val, **'vs.Trigram'**, trigram\_val)

Result:

Part b

[('I', 'PPSS'), ("can't", 'MD\*'), ('tell', 'VB'), ('when', 'WRB'), (',', ','), ('but', 'CC'), ("I'm", 'PPSS+BEM'), ('positive', 'JJ'), ('I', 'PPSS'), ('witnessed', 'VBN'), ('this', 'DT'), ('same', 'AP'), ('scene', 'NN'), ('of', 'IN'), ('this', 'DT'), ('particular', 'JJ'), ('gathering', 'NN'), ('at', 'IN'), ('some', 'DTI'), ('time', 'NN'), ('in', 'IN'), ('the', 'AT'), ('past', 'NN'), ("''", "''"), ('!', '.'), ('!', '.')]

Unigram 0.9382043354880824 vs.Bigram 0.039148670270679684 vs.Trigram 0.039148670270679684

1. Compare the given TrigramTagger from the previous question with a TrigramTagger where no backoff is provided. Train this tagger on all of the sentences from the Brown corpus with the category *news.* Then evaluate your tagger using “evaluate” function on all of the sentences from the Brown corpus with the category *lore.* Report the numbers.Which tagger performs better? Why?

Code:

**def** exercise3():  
 news\_tagged\_sents = brown.tagged\_sents(categories=**'news'**)  
  
  
 t0 = nltk.DefaultTagger(**'NN'**)  
 t1 = nltk.UnigramTagger(news\_tagged\_sents,backoff=t0)  
 t2 = nltk.BigramTagger(news\_tagged\_sents, backoff=t1)  
 t3 = nltk.TrigramTagger(news\_tagged\_sents,backoff=t2)  
  
  
 *# category lore* lore\_tagged\_sents = brown.tagged\_sents(categories=**'lore'**)  
 lore\_trigram\_val = t3.evaluate(lore\_tagged\_sents)  
  
 t4 = nltk.TrigramTagger(news\_tagged\_sents)  
 lore\_trigram\_val\_without = t4.evaluate(lore\_tagged\_sents)  
  
 print(**"Brown corpus category lore value"**, lore\_trigram\_val)  
 print(**"Brown corpus category lore value without"**, lore\_trigram\_val\_without)  
  
 print (**"Category news tagger peforms better because it evaluates tags of the same category,"**)  
 print (**"thus yielding more accurate results. It performs better if evaluate tags in the same category"**)

Result:

Exercise 3

Brown corpus category lore value 0.8427274952628764

Brown corpus category lore value without 0.06240310428925013

Category news tagger peforms better because it evaluates tags of the same category,

thus yielding more accurate results. It performs better if evaluate tags in the same category

1. The majority of WordNet's senses are marked by four POS categories: noun, verb, adjective, and adverb. Determine the percentage of words from the WordNet corpus that have senses in more than one of these categories. For example, *type* has senses which connect to both “noun” and “verb” POS (positive case), whereas *typewriter* has only senses which connect to “noun” POS (negative case).

Code:

**def** exercise4():  
 wn\_words = wn.all\_synsets()*# list of words* lemma\_str = []  
 **for** i **in** wn\_words:  
 **for** lemma **in** i.lemma\_names():  
 lemma\_str.append(lemma)  
  
 lemma\_str\_set = list(set(lemma\_str))  
  
 total\_count = 0  
 **for** i **in** lemma\_str\_set:  
 count = 0  
 **if** len(wn.synsets(i, pos=**'a'**)) > 0:  
 count += 1  
 **if** len(wn.synsets(i, pos=**'n'**)) > 0:  
 count += 1  
 **if** len(wn.synsets(i, pos=**'v'**)) > 0:  
 count += 1  
 **if** len(wn.synsets(i, pos=**'r'**)) > 0:  
 count += 1  
 **if** count > 1:  
 total\_count += 1  
  
 result = total\_count / len(lemma\_str\_set)  
 print(result)

Result:

Exercise 4

0.08274053654272844