**HOMEWORK 5 TEXT MINING WITH PYTHON**

**By- Zeeshan Ali**

[alisz@mail.uc.edu](mailto:alisz@mail.uc.edu)

15. 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? (Only consider regular plurals, formed with the *-s* suffix.)
2. Which word has the greatest number of distinct tags. What are they, and what do they represent?
3. List tags in order of decreasing frequency. What do the 20 most frequent tags represent?
4. Which tags are nouns most commonly found after? What do these tags represent?

**#15.1**

import nltk

from nltk.corpus import brown

#import nltk corpus brown

noun={word for word, pos in brown.tagged\_words() if pos.startswith('NN')}

#get all nouns with tag NN

nouns={word for word, pos in brown.tagged\_words() if pos.startswith('NNS')}

#get all plurals with tag NNS

fnoun=nltk.FreqDist(noun)

#get Frequency distrubtion of nouns

fnouns=nltk.FreqDist(nouns)

#get Frequency distrubtion of nouns plural

#Loop through nouns

for wo in nouns:

#string operation to derive singular from plural

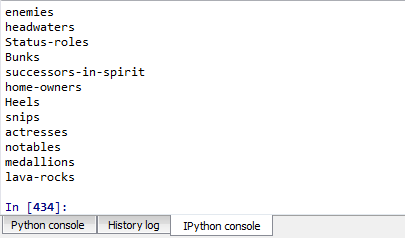
new\_n=wo[:-1]

#if the frequency of plural is more than singular

if(fnouns[wo]>fnoun[new\_n]):

print(wo) #Print the plural if it has more freq than singular

**OUTPUT:**



**#15.2**

#Getting tagged words from brown corpus news category

brown\_news\_tagged = brown.tagged\_words(categories='news', tagset='universal')

#Getting nltk conditional frequency distribution of tags

data = nltk.ConditionalFreqDist((word.lower(), tag) for (word, tag) in brown\_news\_tagged)

z=0 #Initializing a counter

#looping through data variable

for word in sorted(data.conditions()):

#getting the tags for each word

tags = [tag for (tag, \_) in data[word].most\_common()]

#if the number of tags is greater than existing counter

if(z<len(tags)):

z=len(tags) #update the counter with max no. of tags

fr=(word, ' '.join(tags)) #store the word and its tag with most distinct tags

print(fr) #print the words and its tags

**OUTPUT:**



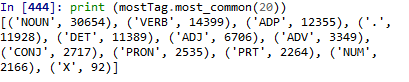
**#15.3**

#Getting Frequency Distribution, from brown news category

mostTag = nltk.FreqDist(t for (w,t) in brown\_news\_tagged)

#Printing most common

print (mostTag.most\_common(20))



**#15.4**

#Noun Bigram

nounBigram =nltk.bigrams(t for (w,t) in brown\_news\_tagged)

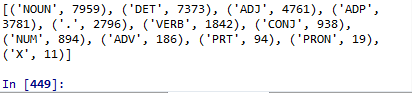
#Freq Distribution,

afterN = nltk.FreqDist(t1 for (t1,t2) in nounBigram if t2 == 'NOUN')

#Printing the most common 20

print (afterN.most\_common(20))

**OUTPUT:**



18. ◑ Generate some statistics for tagged data to answer the following questions:

1. What proportion of word types are always assigned the same part-of-speech tag?
2. How many words are ambiguous, in the sense that they appear with at least two tags?
3. What percentage of word *tokens* in the Brown Corpus involve these ambiguous words?

#18a.

#Calculating CFD for brown corpus

cfd = nltk.ConditionalFreqDist(brown.tagged\_words())

#All conditions tags of from conditional Frequency distribution

cond = cfd.conditions()

#find single tag words

single\_tags = [cond for cond in cond if len(cfd[cond]) == 1]

#Calculating proportion of assigned same part of speech tag or single tag

propSingle= len(single\_tags) / len(cond)

#Printing proportion

print(propSingle)

**OUTPUT:**



#18b.

#Getting tagged words from brown corpus news category

brown\_news\_tagged = brown.tagged\_words(categories='news', tagset='universal')

#Getting nltk conditional frequency distribution of tags

data = nltk.ConditionalFreqDist((word.lower(), tag) for (word, tag) in brown\_news\_tagged)

#looping through data variable

for word in sorted(data.conditions()):

#getting the tags for each word

tags = [tag for (tag, \_) in data[word].most\_common()]

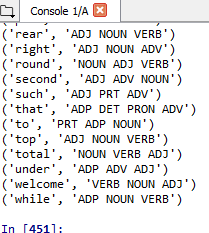
#Checking if the word is coming for more than two tags

if(len(tags)>2):

fr=(word, ' '.join(tags)) #Join all the tags

print(fr) #printing ambiguous word

**OUPUT:**



#18c.

total=len(brown.tagged\_words())

data = nltk.ConditionalFreqDist((word.lower(), tag) for (word, tag) in brown.tagged\_words())

ambiguous=0 #Initializing a counter

#looping through data variable

for word in sorted(data.conditions()):

#getting the tags for each word

tags = [tag for (tag, \_) in data[word].most\_common()]

#Checking if the word is coming for more than two tags

if(len(tags)>2):

ambiguous=ambiguous+1

#Get the percentage of ambiguous words

ambiguous=(ambiguous/total)\*100

#Printing the ambiguous words

print(ambiguous)



21.  In [3.1](http://www.nltk.org/book/ch03.html#tab-absolutely) we saw a table involving frequency counts for the verbs *adore*, *love*, *like*, *prefer* and preceding qualifiers *absolutely* and *definitely*. Investigate the full range of adverbs that appear before these four verbs.

#21.

hal = brown.tagged\_words(tagset='universal')

#getting tagged words

hal\_bigrams = list(nltk.bigrams(hal))

#getting bigrams from tagged words

for bi in hal\_bigrams:

#For each bigrams

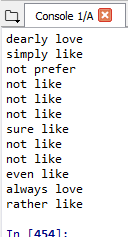
zig = [list(d) for d in zip(\*bi)]

#if the word is adore, love, prefer or like and it is verb which follows adjective

if zig[0][1] in ['adore', 'love', 'prefer', 'like'] and zig[1][1] == 'VERB' and zig[1][0] == 'ADV':

print(zig[0][0],zig[0][1]) #Printing the ADV and VERB word

**OUTPUT:**



22. We defined the regexp\_tagger that can be used as a fall-back tagger for unknown words. This tagger only checks for cardinal numbers. By testing for particular prefix or suffix strings, it should be possible to guess other tags. For example, we could tag any word that ends with *-s* as a plural noun. Define a regular expression tagger (using RegexpTagger()) that tests for at least five other patterns in the spelling of words. (Use inline documentation to explain the rules.)

#22.

import nltk

#importing nltk related packages

from nltk.corpus import brown

patterns = [

(r'.\*ed$', 'VBD'), # verb ending with ed simple past

(r'.\*ing$', 'VBG'), # verb ending in 'ing'

(r'.\*s$', 'NNS'), # plural nouns

(r'.\*\'s', 'NN$'), # possessive, aporstrophe s

(r'.\*', 'NN') # #noun

]

#getting brown sentences from news category

bs = brown.sents(categories='news')

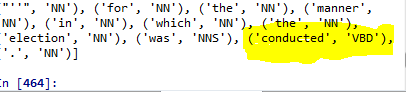
#regular expression for taggers to search for above patters

regexp\_tagger = nltk.RegexpTagger(patterns)

#Printing the matching pattern from brown sentences

print(regexp\_tagger.tag(bs[3]))

**OUTPUT:**



26. ◑ [4.1](http://www.nltk.org/book/ch05.html#code-baseline-tagger) plotted a curve showing change in the performance of a lookup tagger as the model size was increased. Plot the performance curve for a unigram tagger, as the amount of training data is varied.

#26.

def perf(cfd, wl):

# Input arguments conditional freq and worlist

# goes through every word iun the wordlist and returns a dictionary

zz = dict((word, cfd[word].max()) for word in wl)

#Provide the above assigned variable and also there is default noun tagger

bt = nltk.UnigramTagger(model=zz, backoff=nltk.DefaultTagger('NN'))

# bt.evaluate for brown sentences is returned

return bt.evaluate(brown.tagged\_sents(categories='news'))

def disp():

import pylab

#importinh pylab

# pulls in a frequency distribution

wf = nltk.FreqDist(brown.words(categories='news')).most\_common()

# sequentially orders the words by frequency

wfreq = [w for (w, \_) in wf]

# makes a cfd on the words and tags

cfd = nltk.ConditionalFreqDist(brown.tagged\_words(categories='news'))

# returns a list of scales

sizes = 2 \*\* pylab.arange(15)

# for every size, evaluate a baseline tagger based on a training set of that size.

perfs = [perf(cfd, wfreq[:size]) for size in sizes]

#Plotting

pylab.plot(sizes, perfs, '-bo')

# sets all label for axis

pylab.title('Lookup Tagger Performance with Varying Model Size')

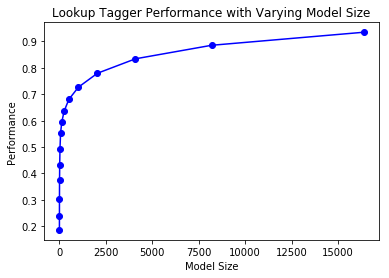
pylab.xlabel('Model Size')

pylab.ylabel('Performance')

pylab.show()

disp() #Calling function

**OUTPUT:**



29. ◑ [4.1](http://www.nltk.org/book/ch05.html#code-baseline-tagger) plotted a curve showing change in the performance of a lookup tagger as the model size was increased. Plot the performance curve for a unigram tagger, as the amount of training data is varied.

#29.

bts = brown.tagged\_sents(categories='news')

#Getting brown sentences

size = int(len(bts) \* 0.9)

#training data 90%

train\_sents = bts[:size]

#testing data

test\_sents = bts[size:]

#Noun as default tagger

t0 = nltk.DefaultTagger('NN')

#Unigran tagger with backoff option as default tagger

t1=nltk.UnigramTagger(train\_sents, backoff=t0)

#Bigram tagger training data

bigram\_tagger = nltk.BigramTagger(train\_sents)

#bigram seen data

bigram\_tagger.tag(bts[2007])

#bigram unseen data

unseen\_sent = bts[4203]

#bigram unseen data

bigram\_tagger.tag(unseen\_sent)

#bigram training data

bitag1=nltk.BigramTagger(train\_sents)

**#Bigram with cutoff=2 option will ignore the words unless it is atleast 2 times,**

**#so those data that are in training data and seen once only will be not recognized**

bitag2=nltk.BigramTagger(train\_sents, cutoff=2)

#Evaluate with cutoff=2

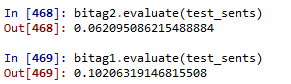
bitag2.evaluate(test\_sents)

#Evaluate without cutoff=2

bitag1.evaluate(test\_sents)

When cutoff option is used, then it will fail to recognize those words who it sees once even if it has already seen it in training data.

**OUTPUT:**



30.  Preprocess the Brown News data by replacing low frequency words with *UNK*, but leaving the tags untouched. Now train and evaluate a bigram tagger on this data. How much does this help? What is the contribution of the unigram tagger and default tagger now?

#30

#Get sentences from brown

sentences = brown.tagged\_sents(categories="news")

#Frequency ditribution of words

freqW = nltk.FreqDist(brown.words())

#Counter for sentences

n1 = []

#For loop of sentences

for t in sentences:

n2 = [] #couter for word

#Couter for each word

for wad in t:

#if word freq is 1 then assign UNK

if freqW[wad]==1:

n2.append(("UNK", wad[1]))

else:

#else assign normal word and its tag

n2.append((wad[0], wad[1]))

#Append the n2 words in n1 sentence counter

n1.append(n2)

#Training data

size = int(len(n1) \* 0.9)

train = n1[:size]

#testing data

test = n1[size:]

t0 = nltk.DefaultTagger('NN') #Noun default tagger

t1 = nltk.UnigramTagger(train, backoff=t0) #unigram tagger

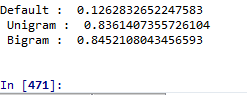
t2 = nltk.BigramTagger(train, backoff=t1) #bigram tagger

#Printing the evaluation

print("Default : ",t0.evaluate(test),"\n","Unigram : " ,t1.evaluate(test),"\n","Bigram : ", t2.evaluate(test),"\n")

**OUTPUT:**

Performances for default, unigram and bigram



31.  Modify the program in [4.1](http://www.nltk.org/book/ch05.html#code-baseline-tagger) to use a logarithmic scale on the x-axis, by replacing pylab.plot() with pylab.semilogx(). What do you notice about the shape of the resulting plot? Does the gradient tell you anything?

#31

#Define perf function which taked cond freq dist and word list as input arg

def perf2(cfd, wl):

#assign word and cfd as dictionary to variable

zz = dict((word, cfd[word].max()) for word in wl)

bt = nltk.UnigramTagger(model=zz, backoff=nltk.DefaultTagger('NN'))

return bt.evaluate(brown.tagged\_sents(categories='news'))

#define disp function to plot

def disp2():

import pylab #import pylab

word\_freqs = nltk.FreqDist(brown.words(categories='news')).most\_common() #word freq

words\_by\_freq = [w for (w, \_) in word\_freqs] #each word freq

cfd = nltk.ConditionalFreqDist(brown.tagged\_words(categories='news')) #conditional freq for news category

sizes = 2 \*\* pylab.arange(15) #pylabe arrange labels distance

perfs = [perf2(cfd, words\_by\_freq[:size]) for size in sizes] #loop through sizes

pylab.semilogx(sizes, perfs, '-bo')

#Assign label of axis

pylab.title('Lookup Tagger Performance with Varying Model Size')

pylab.xlabel('Model Size')

pylab.ylabel('Performance')

pylab.show() #Plot diagram

disp2() #Call disp function

**OUTPUT:**

