# Homework 2 Python Text Mining

By- Zeeshan Ali

**Q13** What percentage of noun synsets have no hyponyms? You can get all noun synsets using wn.all\_synsets('n').

import nltk

from nltk.corpus import wordnet as wn

#importing nltk and wordnet

an=wn.all\_synsets('n') #importing all nouns

anum=len(set(an)) #counting all nouns

anum

#Nouns with No hyponyms or Hyponyms as 0

nn=[w for w in wn.all\_synsets('n') if len(w.hyponyms()) ==0]

nnum=len(nn)

nnum

percentage=(nnum)/anum\*100 #Calculating % of no hyponyms with all nouns

percentage

print("Percentage is "+str(percentage))

**OUTPUT**



**Q14** Define a function supergloss(s) that takes a synset s as its argument and returns a string consisting of the concatenation of the definition of s, and the definitions of all the hypernyms and hyponyms of s.

#Q14

#Function Supergloss accepting sysnet argument and return concatonated string

def supergloss(s):

s\_def=s.definition()

hypo\_def=s.hyponyms()

listed=[]

for w in hypo\_def:

listed.append(w.definition())

hyper\_def=s.hypernyms()

for w in hyper\_def:

listed.append(w.definition())

return s\_def+str(listed)

#Calling functions

supergloss(wn.synset('car.n.01'))

**OUTPUT:**

Out[227]: 'a motor vehicle with four wheels; usually propelled by an internal combustion engine[\'a vehicle that takes people to and from hospitals\', \'a car that has a long body and rear door with space behind rear seat\', \'a car that is old and unreliable\', \'a car driven by a person whose job is to take passengers where they want to go in exchange for money\', \'a small and economical car\', \'a car that has top that can be folded or removed\', \'a car with two doors and front seats and a luggage compartment\', \'a car in which policemen cruise the streets; equipped with radiotelephonic communications to headquarters\', \'a car that is powered by electricity\', \'a car with relatively low fuel efficiency\', \'a car that resembles a convertible but has a fixed rigid top\', \'a car having a hatchback door\', \'an early term for an automobile\', \'a car modified to increase its speed and acceleration\', \'a car suitable for traveling over rough terrain\', \'large luxurious car; usually driven by a chauffeur\', \'a car that is lent as a replacement for one that is under repair\', \'a car that is even smaller than a subcompact car\', \'a small box-shaped passenger van; usually has removable seats; used as a family car\', \'the first widely available automobile powered by a gasoline engine; mass-produced by Henry Ford from 1908 to 1927\', \'a high-performance car that leads a parade of competing cars through the pace lap and then pulls off the course\', \'a fast car that competes in races\', \'an open automobile having a front seat and a rumble seat\', \'a car that is closed and that has front and rear seats and two or four doors\', \'a high-performance four-wheel drive car built on a truck chassis\', \'a small low car with a high-powered engine; usually seats two persons\', \'a steam-powered automobile\', "a car kept in dealers\' stock for regular sales", \'a car smaller than a compact car\', \'large open car seating four with folding top\', \'a car that has been previously owned; not a new car\', \'a self-propelled wheeled vehicle that does not run on rails\']'

**Q16** Write a program to generate a table of lexical diversity scores (i.e. token/type ratios), as we saw in 1.1. Include the full set of Brown Corpus genres (nltk.corpus.brown.categories()). Which genre has the lowest diversity (greatest number of tokens per type)? Is this what you would have expected?

**#Q16**

**#Getting the lexical diversity of the categories in brown corpus**

**from nltk.corpus import brown**

**for z in nltk.corpus.brown.categories():**

**l1 = len(set(w.lower() for w in brown.words(categories=z)))**

**l2= len(brown.words(categories=z))**

**diversity = l1/l2**

**print (z, diversity)**

**OUTPUT:**

adventure 0.11953794237258804

belles\_lettres 0.09854647132227204

editorial 0.1478637750795403

fiction 0.1267375306623058

government 0.10498167348859763

hobbies 0.1314469609569494

**humor 0.21917492509794884**

**learned 0.085085327234342**

lore 0.12151515426250464

mystery 0.11305077926848467

news 0.13039759731089762

religion 0.15053681565521967

reviews 0.19823604559748428

romance 0.11257890377309988

science\_fiction 0.20953697304768487

Learned is least diverse whereas humor is most diverse, the diversity is solely based upon the sample present in brown corpus.

**Q17** Write a function that finds the 50 most frequently occurring words of a text that are not stopwords.

#Q17

#Function to 50 most common words that are not stop words

from nltk.book import \*

def common\_words(text):

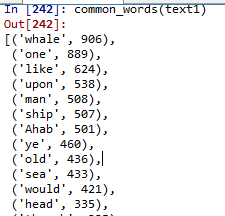
stopwords = nltk.corpus.stopwords.words('english')

ct = [w for w in text if w.lower() not in stopwords and w.isalpha()]

return nltk.FreqDist(ct).most\_common(50)

common\_words(text1)

**OUTPUT:**



**Q-18** Write a program to print the 50 most frequent bigrams (pairs of adjacent words) of a text, omitting bigrams that contain stopwords.

#Q18

#Function to avoid bigrams of stop words and print common bigrams

def common\_bigrams(text):

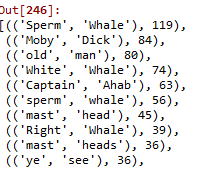
sw = nltk.corpus.stopwords.words('english')

ct = [w for w in text if w.lower() not in sw and w.isalpha()]

return FreqDist(nltk.bigrams(ct)).most\_common(50)

common\_bigrams(text1)

**OUTPUT:**



**Q20** Write a function word\_freq() that takes a word and the name of a section of the Brown Corpus as arguments, and computes the frequency of the word in that section of the corpus.

**#Q20**

**#Function to check word freq**

**def word\_freq(word, se):**

**from nltk.corpus import brown**

**freq=0**

**for w in brown.words(categories=se):**

**if w==word:**

**freq+=1**

**return "Frequency "+word+" in "+se+" is "+str(freq)**

**#Call**

**word\_freq("love", "belles\_lettres")**

**OUTPUT:**



**Q22** Define a function hedge(text) which processes a text and produces a new version with the word 'like' between every third word.

**#Q22**

**def hedge(text):**

**counter=0**

**list1=[]**

**for w in text:**

**counter+=1**

**if counter%3==0:**

**list1.append("like")**

**list1.append(w)**

**else:**

**list1.append(w)**

**return (list1)**

**hedge(text1)**

**OUTPUT:**

Last portion of the returned text

