PYTHON TEXT MINING HOMEWORK 4

By – Syed Zeeshan Ali

M10614655 | [alisz@mail.uc.edu](mailto:alisz@mail.uc.edu)

13. Write code to initialize a two-dimensional array of sets called word\_vowels and process a list of words, adding each word to word\_vowels[l][v] where l is the length of the word and v is the number of vowels it contains.

#13

#Creating a word list of different words

word\_list=['aaa','finish','loper','farhenieit']

#Declaring Viwels both in small and capital

f="aeiouAEIOU"

#Getting the maximum length out of all the words

dd=len(max(word\_list, key=len))

#Declaring a two dimensional matrix of [max\_length] X [max\_length] size of the word

word\_vowels = [['' for z in range(dd)] for j in range(dd)]

#Initializing the matrix with ' '

print(word\_vowels)

#Looping each word

for i in word\_list:

count=0 #Initializing counter

l=len(i) #length of word

for v in i: #Looping letter of word

if v in f: #if letter in vowels

count=count+1 #Increase Counter

print(l,count,i) #Printing value for debugging

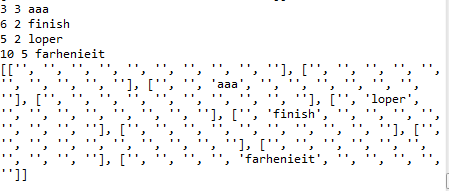
word\_vowels[l-1][count-1]=i #we are doing -1 because the index are 0 to n-1

#Assigning to matrix, row is length of word and column is sum of vowel in word

#Printing matrix

print(word\_vowels)

**OUTPUT:**



“aaa” 3 Letters and 3 vowels so it goes to 3rd row and 3rd Col same thing happen with other words in the list.

15. Write a program that takes a sentence expressed as a single string, splits it and counts up the words. Get it to print out each word and the word's frequency, one per line, in alphabetical order.

#15

import nltk

#import nltk

#Declare empty list

li=[]

#Declare empty string

wor=''

#Check until user enters something

while(len(wor)==0):

wor=input("Enter a Sentence : ") #User input from terminal

if(len(wor)!=0): #Check the length of the string entered

break #if the length is greater than 0 for wor variable then break

li=wor.split(' ') #split on the basis of space

x=len(li) #number of words in list

z=nltk.FreqDist(li) #Frequency Distribution

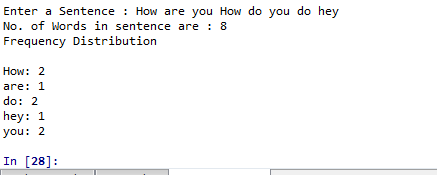
print("No. of Words in sentence are :",x) #Printing no. of words

print("Frequency Distribution\n")

for key in sorted(z): #Printing sorted

print("%s: %s"%(key, z[key])) #Key and value pairs

**OUTPUT:**



**19.** Write a list comprehension that sorts a list of WordNet synsets for proximity to a given synset. For example, given the synsets minke\_whale.n.01, orca.n.01, novel.n.01, and tortoise.n.01, sort them according to their shortest\_path\_distance() from right\_whale.n.01.

#19

from nltk.corpus import wordnet as wn

#Importing nltk corpus wordnet

#Defining a function that returns the shortest path distance for each synset

def funcNew(a, b):

disl = [(a.shortest\_path\_distance(syn), syn) for syn in b] #Calling shortest Distance

disl.sort() #sorting the list

return disl #returning the value

#Other Synset from which we want to compare distance

warp=['minke\_whale.n.01', 'orca.n.01', 'novel.n.01', 'tortoise.n.01']

#Synsets for each string

b = [wn.synset(syn) for syn in warp]

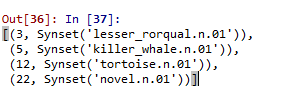
#Synset for 'right\_whale.n.01'

a=wn.synset('right\_whale.n.01')

#Calling the funcion which calculates the shortest distance and returns in sorted manner

funcNew(a, b)

**OUTPUT:**



**20.** Write a function that takes a list of words (containing duplicates) and returns a list of words (with no duplicates) sorted by decreasing frequency. E.g. if the input list contained 10 instances of the word table and 9 instances of the word chair, then table would appear before chair in the output list.

#Define a variable dupC

def dupC(st):

print(st) #Debugging print

nw=st.split(" ") #Splitting sentence on space

nn=nltk.FreqDist(nw) #Frequency Distribution

length = len(set(nn)) #len of set

j = list(nn.most\_common(length)) #Getting the most\_common

j = [i[0] for i in j] #Picking first element

print(j) #Printing the unique output

dupC("Z Z Z sha Na N Na p pop p")

**OUTPUT:**



21. Write a function that takes a text and a vocabulary as its arguments and returns the set of words that appear in the text but not in the vocabulary. Both arguments can be represented as lists of strings. Can you do this in a single line, using set.difference()?

#21

#21

#Words

words=['aaa','bbb','ccc']

#Vocablary list

vocab=['aaa','bbb','ddd','eee']

#Defining function which takes set difference and prints it

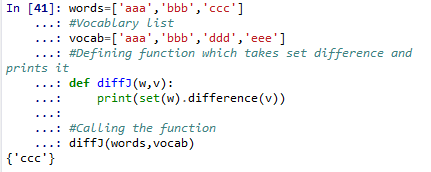
def diffJ(w,v):

print(set(w).difference(v))

#Calling the function

diffJ(words,vocab)

**OUTPUT:**



22. Import the itemgetter() function from the operator module in Python's standard library (i.e. from operator import itemgetter). Create a list words containing several words. Now try calling:sorted(words, key=itemgetter(1)), and sorted(words, key=itemgetter(-1)). Explain what itemgetter() is doing.

#22

#

from operator import itemgetter

#words=['zss','eisos','voldo','poper','harakiri','ain','nebula']

words=['boil','afc','sack','jazz','fa']

#It is part of operator package

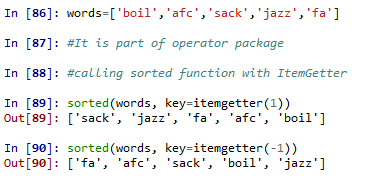
#calling sorted function with ItemGetter

sorted(words, key=itemgetter(1))

sorted(words, key=itemgetter(-1))

itemgetter(1) makes the sorting based on first element

itemgetter(-1) makes the sorting based on last letter lexicographically.



23. Write a recursive function lookup(trie, key) that looks up a key in a trie, and returns the value it finds. Extend the function to return a word when it is uniquely determined by its prefix (e.g. vanguard is the only word that starts with vang-, so lookup(trie, 'vang') should return the same thing as lookup(trie, 'vanguard')).

#23

#list of words to test from

words=['z','shan','jeep','vanguard','pop','faught']

#function lookup arguments trie and key

def lookup(trie,key):

z=[x for x in trie if key in x]

print(z)

#calling lookup function

lookup(words,'van')

**OUTPUT:**

