Report on

CE887-7-AU – Natural Language Engineering

Assignment 1: Probabilities, Regular Expressions & Language Models Aline Villavicencio October 2018

Student Name : Atiwat Onsuwan (ao18409)

Part 1: Tokenization, Part-of-Speech Tagging (30%)

Q1 (20%) Create a program that reads the text from the following website and identify all the types and tokens before and after lowercasing and lemmatization. Use NLTK functions to perform these tasks, from the url reader to the tokenizer and lemmatizer.
https://www.theguardian.com/music/2018/oct/19/while-my-guitar-gently-weeps-beatles-george-harrison

The output of the program should contain the following information: This text contains types before lemmatization: This text contains tokens before lemmatization: This text contains types after lemmatization: This text contains tokens after lemmatization:

Answer.1

```
import re
import nltk
from nltk import word_tokenize
from bs4 import BeautifulSoup
import urllib
from urllib import request
from nltk.stem.wordnet import WordNetLemmatizer
from nltk import wordnet

source=urllib.request.urlopen("https://www.theguardian.com/music/2018/oct/19/while-my-guitar-gently-weeps-beatles-george-harrison").read().decode("utf-8")
soup=BeautifulSoup(source,'html5lib')

Text = ''
for para in soup.find_all('p'):
    Text +=(para.text)#Original Text

print(Text) #DISPLAY ORIGINAL TEXT EXTRACT FROM URL
lowertext=Text.lower() # convert MAIN TEXT  into lower

# remove punctuation  Before LOWERING
mytext_tokens_nopunct = [word for word in word_tokenize(Text)
if re.search("\w", word)]

#Removing Punctuation After LOWRING
mylowertext_tokens_nopunct = [word for word in word_tokenize(lowertext)
if re.search("\w", word)]


print('TYPES BEFORE LOWERING AND LEMMATIZER  ')
print(len(set(mytext_tokens_nopunct)))
print('tokens BEFORE LOWER AND LEMMATIZER ')
print(len(mytext_tokens_nopunct))
```

```
print('TYPES AFTER LOWER  ')
print(len(set(mylowertext_tokens_nopunct)))
print('tokens AFTER ')
print(len(mylowertext_tokens_nopunct))

#NOW LAMMATIZING
wordnet_lemmatizer = WordNetLemmatizer()
nltk_tokens = nltk.word_tokenize(lowertext)
mylemma_tokens_nopunct = [word for word in word_tokenize(lowertext)
if re.search("\w", word)]
w1=[]
for w in nltk_tokens:
    w1.append(wordnet_lemmatizer.lemmatize(w))
#print ((w1))
print('TYPES AFTER  LEMMATIZER  ')
print(len(set(mylemma_tokens_nopunct)))
print('TOKENS AFTER  LEMMATIZER ')
print(len(mylemma_tokens_nopunct))
```

Acoustic demos of the song, regarded as one of George Harrison's best compositions, to be included in remastered White Album set

Ben Beaumont-Thomas

Fri 19 Oct 2018 12.09 BST

Last modified on Fri 19 Oct 2018 17.30 BST

Three unheard versions of While My Guitar Gently Weeps, regarded by many as George Harrison's greatest contribution to the Beatles, have been released online.The song was written by Harrison in 1968, after he had studied transcendental meditation with the Maharishi Mahesh Yogi in India. There are two completely unheard versions: The Esher Demo is an acoustic version, complete with beautiful multitracked vocals, recorded at Harrison's house in Surrey, in May 1968, in preparation for the studio recordings that began later that month. It is one of 27 demos recorded at the house included on a forthcoming remastered version of The White Album.Another, titled Acoustic Version Take 2, is a raw, shaky but endearing alternative to the solo Harrison demo, played on acoustic guitar and harmonium, that was previously released as part of the Anthology 3 compilation in 1996. At one point Harrison complains, "Yeah, maybe you'll have to give him his own mic" to someone in the studio, as the harmonium drifts into discordance.The final unheard version is a newly remastered mix of the album track. Like the original, it features an electric guitar part from Eric Clapton – he once said he felt the song was about Harrison's dislocation from the group, who didn't share Harrison's zeal for eastern mysticism following their trip to India. Ringo Starr infamously brought a suitcase full of Heinz beans on the trip because he didn't want to eat spicy food, and left after two weeks.The newly remastered White Album is released 9 November, and, as well as the Esher demos, also features 50 mostly unreleased additional recordings made during the studio sessions.

TYPES BEFORE LOWERING AND LEMMATIZER

189

tokens BEFORE LOWER AND LEMMATIZER

314

TYPES AFTER LOWER

181

tokens AFTER

314

TYPES AFTER LEMMATIZER

181

TOKENS AFTER LEMMATIZER

314

Q.2

a) Assign part-of-speech (POS) tags to all tokens in the text used above. Use one of the implemented POS taggers in NLTK to do this. (5%)

b)POS taggers do not always assign correct tags to words. Identify tagging errors in the sentences above and briefly explain why these errors may have been caused.

Answer.2-a

Acoustic demos of the song, regarded as one of George Harrison's best compositions, to be included in remastered White Album set

Ben Beaumont-Thomas

Fri 19 Oct 2018 12.09 BST

Last modified on Fri 19 Oct 2018 17.30 BST

Three unheard versions of While My Guitar Gently Weeps, regarded by many as George Harrison's greatest contribution to the Beatles, have been released online.The song was written by Harrison in 1968, after he had studied transcendental meditation with the Maharishi Mahesh Yogi in India. There are two completely unheard versions: The Esher Demo is an acoustic version, complete with beautiful multitracked vocals, recorded at Harrison's house in Surrey, in May 1968, in preparation for the studio recordings that began later that month. It is one of 27 demos recorded at the house included on a forthcoming remastered version of The White Album.Another, titled Acoustic Version Take 2, is a raw, shaky but endearing alternative to the solo Harrison demo, played on acoustic guitar and harmonium, that was previously released as part of the Anthology 3 compilation in 1996. At one point Harrison complains, "Yeah, maybe you'll have to give him his own mic" to someone in the studio, as the harmonium drifts into discordance.The final unheard version is a newly remastered mix of the album track. Like the original, it features an electric guitar part from Eric Clapton – he once said he felt the song was about Harrison's dislocation from the group, who didn't share Harrison's zeal for eastern

mysticism following their trip to India. Ringo Starr infamously brought a suitcase full of Heinz beans on the trip because he didn't want to eat spicy food, and left after two weeks.The newly remastered White Album is released 9 November, and, as well as the Esher demos, also features 50 mostly unreleased additional recordings made during the studio sessions.

[('Acoustic', 'ADJ'), ('demos', 'NOUN'), ('of', 'ADP'), ('the', 'DET'), ('song', 'NOUN'), (',', '.'), ('regarded', 'VERB'), ('as', 'ADP'), ('one', 'NUM'), ('of', 'ADP'), ('George', 'NOUN'), ('Harrison', 'NOUN'), ('’', 'NOUN'), ('s', 'NOUN'), ('best', 'ADJ'), ('compositions', 'NOUN'), (',', '.'), ('to', 'PRT'), ('be', 'VERB'), ('included', 'VERB'), ('in', 'ADP'), ('remastered', 'ADJ'), ('White', 'NOUN'), ('Album', 'NOUN'), ('set', 'VERB'), ('Ben', 'NOUN'), ('Beaumont-Thomas', 'NOUN'), ('Fri', 'NOUN'), ('19', 'NUM'), ('Oct', 'NOUN'), ('2018', 'NUM'), ('12.09', 'NUM'), ('BST', 'NOUN'), ('Last', 'ADJ'), ('modified', 'VERB'), ('on', 'ADP'), ('Fri', 'NOUN'), ('19', 'NUM'), ('Oct', 'NOUN'), ('2018', 'NUM'), ('17.30', 'NUM'), ('BST', 'NOUN'), ('Three', 'NOUN'), ('unheard', 'ADJ'), ('versions', 'NOUN'), ('of', 'ADP'), ('While', 'ADP'), ('My', 'NOUN'), ('Guitar', 'NOUN'), ('Gently', 'NOUN'), ('Weeps', 'NOUN'), (',', '.'), ('regarded', 'VERB'), ('by', 'ADP'), ('many', 'ADJ'), ('as', 'ADP'), ('George', 'NOUN'), ('Harrison', 'NOUN'), ('’', 'NOUN'), ('s', 'VERB'), ('greatest', 'ADJ'), ('contribution', 'NOUN'), ('to', 'PRT'), ('the', 'DET'), ('Beatles', 'NOUN'), (',', '.'), ('have', 'VERB'), ('been', 'VERB'), ('released', 'VERB'), ('online.The', 'NOUN'), ('song', 'NOUN'), ('was', 'VERB'), ('written', 'VERB'), ('by', 'ADP'), ('Harrison', 'NOUN'), ('in', 'ADP'), ('1968', 'NUM'), (',', '.'), ('after', 'ADP'), ('he', 'PRON'), ('had', 'VERB'), ('studied', 'VERB'), ('transcendental', 'ADJ'), ('meditation', 'NOUN'), ('with', 'ADP'), ('the', 'DET'), ('Maharishi', 'NOUN'), ('Mahesh', 'NOUN'), ('Yogi', 'NOUN'), ('in', 'ADP'), ('India', 'NOUN'), ('.', '.'), ('There', 'DET'), ('are', 'VERB'), ('two', 'NUM'), ('completely', 'ADV'), ('unheard', 'ADJ'), ('versions', 'NOUN'), (':', '.'), ('The', 'DET'), ('Esher', 'NOUN'), ('Demo', 'NOUN'), ('is', 'VERB'), ('an', 'DET'), ('acoustic', 'ADJ'), ('version', 'NOUN'), (',', '.'), ('complete', 'ADJ'), ('with', 'ADP'), ('beautiful', 'ADJ'), ('multitracked', 'ADJ'), ('vocals', 'NOUN'), (',', '.'), ('recorded', 'VERB'), ('at', 'ADP'), ('Harrison', 'NOUN'), ('’', 'NOUN'), ('s', 'NOUN'), ('house', 'NOUN'), ('in', 'ADP'), ('Surrey', 'NOUN'), (',', '.'), ('in', 'ADP'), ('May', 'NOUN'), ('1968', 'NUM'), (',', '.'), ('in', 'ADP'), ('preparation', 'NOUN'), ('for', 'ADP'), ('the', 'DET'), ('studio', 'NOUN'), ('recordings', 'NOUN'), ('that', 'DET'), ('began', 'VERB'), ('later', 'ADV'), ('that', 'ADP'), ('month', 'NOUN'), ('.', '.'), ('It', 'PRON'), ('is', 'VERB'), ('one', 'NUM'), ('of', 'ADP'), ('27', 'NUM'), ('demos', 'NOUN'), ('recorded', 'VERB'), ('at', 'ADP'), ('the', 'DET'), ('house', 'NOUN'), ('included', 'VERB'), ('on', 'ADP'), ('a', 'DET'), ('forthcoming', 'NOUN'), ('remastered', 'ADJ'), ('version', 'NOUN'), ('of', 'ADP'), ('The', 'DET'), ('White', 'NOUN'), ('Album.Another', 'NOUN'), (',', '.'), ('titled', 'VERB'), ('Acoustic', 'ADJ'), ('Version', 'NOUN'), ('Take', 'NOUN'), ('2', 'NUM'), (',', '.'), ('is', 'VERB'), ('a', 'DET'), ('raw', 'ADJ'), (',', '.'), ('shaky', 'ADJ'), ('but', 'CONJ'), ('endearing', 'VERB'), ('alternative', 'ADJ'), ('to', 'PRT'), ('the', 'DET'), ('solo', 'NOUN'), ('Harrison', 'NOUN'), ('demo', 'NOUN'), (',', '.'), ('played', 'VERB'), ('on', 'ADP'), ('acoustic', 'ADJ'), ('guitar', 'NOUN'), ('and', 'CONJ'), ('harmonium', 'NOUN'), (',', '.'), ('that', 'DET'), ('was', 'VERB'), ('previously', 'ADV'), ('released', 'VERB'), ('as', 'ADP'), ('part', 'NOUN'), ('of', 'ADP'), ('the', 'DET'), ('Anthology', 'NOUN'), ('3', 'NUM'), ('compilation', 'NOUN'), ('in', 'ADP'), ('1996', 'NUM'), ('.', '.'), ('At', 'ADP'), ('one', 'NUM'), ('point', 'NOUN'), ('Harrison', 'NOUN'), ('complains', 'VERB'), (',', '.'), ('“', 'NOUN'), ('Yeah', 'NOUN'), (',', '.'), ('maybe', 'ADV'), ('you', 'PRON'), ('’', 'VERB'), ('ll', 'NOUN'), ('have', 'VERB'), ('to', 'PRT'), ('give', 'VERB'), ('him', 'PRON'), ('his', 'PRON'), ('own', 'ADJ'), ('mic', 'ADJ'), ('”', 'NOUN'), ('to', 'PRT'), ('someone', 'NOUN'), ('in', 'ADP'), ('the', 'DET'), ('studio', 'NOUN'), (',', '.'), ('as', 'ADP'), ('the', 'DET'), ('harmonium', 'NOUN'), ('drifts', 'VERB'), ('into', 'ADP'), ('discordance.The', 'ADJ'), ('final', 'ADJ'), ('unheard', 'ADJ'), ('version', 'NOUN'), ('is', 'VERB'), ('a', 'DET'), ('newly', 'ADV'), ('remastered', 'VERB'), ('mix', 'NOUN'), ('of', 'ADP'), ('the', 'DET'), ('album', 'ADJ'), ('track', 'NOUN'), ('.', '.'), ('Like', 'ADP'), ('the', 'DET'), ('original', 'ADJ'), (',', '.'), ('it', 'PRON'), ('features', 'VERB'), ('an', 'DET'), ('electric', 'ADJ'), ('guitar',

'NOUN'), ('part', 'NOUN'), ('from', 'ADP'), ('Eric', 'NOUN'), ('Clapton', 'NOUN'), ('–', 'NOUN'), ('he', 'PRON'), ('once', 'ADV'), ('said', 'VERB'), ('he', 'PRON'), ('felt', 'VERB'), ('the', 'DET'), ('song', 'NOUN'), ('was', 'VERB'), ('about', 'ADP'), ('Harrison', 'NOUN'), ('’', 'NOUN'), ('s', 'ADJ'), ('dislocation', 'NOUN'), ('from', 'ADP'), ('the', 'DET'), ('group', 'NOUN'), (',', '.'), ('who', 'PRON'), ('didn', 'VERB'), ('’', 'NOUN'), ('t', 'NOUN'), ('share', 'NOUN'), ('Harrison', 'NOUN'), ('’', 'NOUN'), ('s', 'VERB'), ('zeal', 'NOUN'), ('for', 'ADP'), ('eastern', 'ADJ'), ('mysticism', 'NOUN'), ('following', 'VERB'), ('their', 'PRON'), ('trip', 'NOUN'), ('to', 'PRT'), ('India', 'NOUN'), ('.', '.'), ('Ringo', 'NOUN'), ('Starr', 'NOUN'), ('infamously', 'ADV'), ('brought', 'VERB'), ('a', 'DET'), ('suitcase', 'NOUN'), ('full', 'ADJ'), ('of', 'ADP'), ('Heinz', 'NOUN'), ('beans', 'NOUN'), ('on', 'ADP'), ('the', 'DET'), ('trip', 'NOUN'), ('because', 'ADP'), ('he', 'PRON'), ('didn', 'VERB'), ('’', 'ADJ'), ('t', 'NOUN'), ('want', 'VERB'), ('to', 'PRT'), ('eat', 'VERB'), ('spicy', 'NOUN'), ('food', 'NOUN'), (',', '.'), ('and', 'CONJ'), ('left', 'VERB'), ('after', 'ADP'), ('two', 'NUM'), ('weeks.The', 'ADJ'), ('newly', 'ADV'), ('remastered', 'VERB'), ('White', 'NOUN'), ('Album', 'NOUN'), ('is', 'VERB'), ('released', 'VERB'), ('9', 'NUM'), ('November', 'NOUN'), (',', '.'), ('and', 'CONJ'), (',', '.'), ('as', 'ADV'), ('well', 'ADV'), ('as', 'ADP'), ('the', 'DET'), ('Esher', 'NOUN'), ('demos', 'NOUN'), (',', '.'), ('also', 'ADV'), ('features', 'VERB'), ('50', 'NUM'), ('mostly', 'ADV'), ('unreleased', 'ADJ'), ('additional', 'ADJ'), ('recordings', 'NOUN'), ('made', 'VERB'), ('during', 'ADP'), ('the', 'DET'), ('studio', 'NOUN'), ('sessions', 'NOUN'), ('.', '.')]

Answer.2-b

1.(‘gently’, ‘NOUN’) = Adverb

2.(‘forthcoming’,’NOUN’) = Adjective

3.(‘because’,’ADP’) = conjunction

Part 2: Regular Expressions, FSAs, and FSTs (30%)

In this part of the assignment you will do some simple information extraction, namely the identification of telephone numbers in text.

Q.3 (20%) Write a regular expression that can find all telephone numbers in a text. Your expression should be able to deal with different formats, for example +55 51 33083838, 1206 872020, 01206 872020 and 05679401945 as well as +44 5679401945 and 0044 5679401945. For full marks: include the output of a Python program that applies your regular expression to any url specified by the user, reads it and finds the telephone numbers. The output should clearly identify what the telephone number is:

Found a match!

Telephone: 01206872020

Answer.3

```
import nltk
import re

text =(" so I need to match  +55 51 33083838, 1206 872020,01206 872020 ,05679401945 ,
+44 5679401945 , 0044 5679401945 Can y" )
```

```
print('Found a  matchs')
print('Telephones:')
phone_regex = re.compile("([\+|0|1])\s*(\d{1,10})\s*(\d{1,})\s*(\d{1,})")
groups = phone_regex.findall(text)
i=1
for g in groups:
    print("".join(g))
    i+=1
```

Output.3

Found a match!

Telephones:

+555133083838

1206872020

01206872020

05679401945

+445679401945

00445679401945

Q.4 (10%) Write a FSA equivalent to the regular expression you just wrote. You can either use a drawing program or write down a transition table.

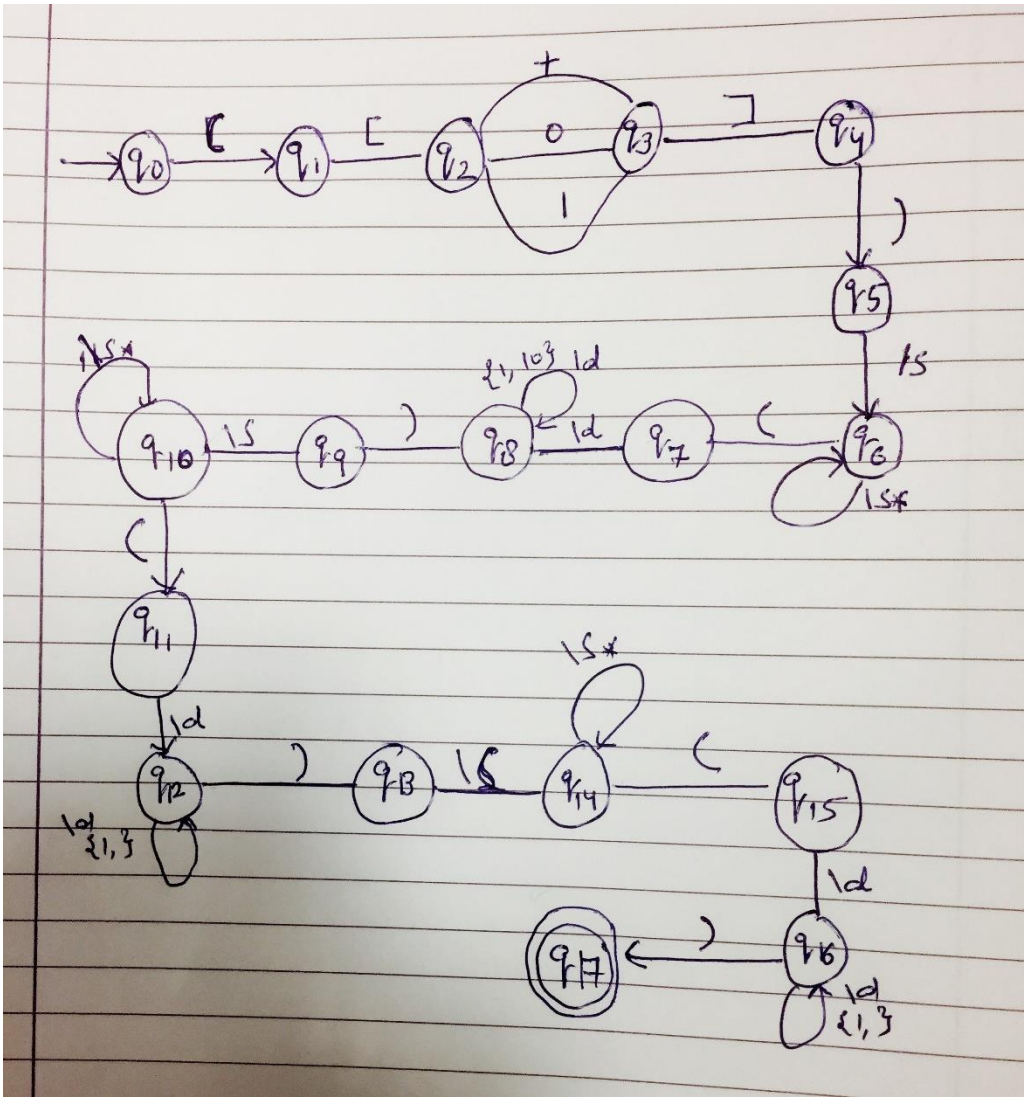
Answer.4

Regular Expressions: ([\+|0|1])\s*(\d{1,10})\s*(\d{1,})\s*(\d{1,})

State Transition Table

State's	q0	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14	q15	q16	q17
q0		(																
q1			[															
q2				+|0|1														
q3					]													
q4						)												
q5							\s											
q6							\s*	(										
q7									\d									
q8									{1,10}	)								
q9											\s							
q10											\s*	(						
q11													\d					
q12													{1, }	)				
q13															\s			
q14															\s*	(		
q15																	\d	
q16																	{1, }	)
*q17																		FINAL

Finite State Automata



Part 3: N-gram models (40%)

Q.5 (10%) Computing a unigram model

a) Use the Toy dataset. The vocabulary is the words in the sampledata.vocab.txt plus the UNK token. Do not include and in the vocabulary. a) Compute the probabilities in a unigram language model without smoothing. Show your work for P(a), P(c), P(UNK). Create a table in the following format and list all of the probabilities in the unigram model.

b) Smooth the model using Laplace smoothing. Show your work for P(a), P(c), P(UNK). Show all the smoothed probabilities in a table.

Program for questions No .5,6 and 7

```
import nltk #import nltk package
import collections # import collections package
from collections import Counter, defaultdict # import counter and defaultdict to count
the number of occurrences
import re # import regular expression package
from tabulate import tabulate #import package 'tabulate' to display table

# this function reads the files and split words by space (\s+) and define number of
array by \n
def read_file(file_path):
    with open(file_path) as f:
        return [re.split("\s+", line.rstrip('\n')) for line in f]

# this function needs 2 parameters from read file ## to remove <s> and </s> from the
data set
def remove_cover(text, cond):
    if cond == "model": #do if cond given = "model"
        for a in (text):
            del a[0]
            del a[-1]
        remove_cover.data = []
        for a in (text):
            remove_cover.data += a
    if cond == "prob":#do if cond given = "prob"
        for a in (text):
            del a[0]
            del a[-1]

# this function is used to calculate UNIGRAM probability of each word and UNK ## 2
parameters are used in this function 1) data set without <s>,</s> 2) number of UNK
found
def unigram_model(data_set, unk):
    data_occurrence = collections.Counter(data_set) # 'data_occurrence' is stored
number of occurrences of each word in data set
    data_index = list(data_occurrence) # 'data_index' stores list of 'data_occurrence'
    letter_count = 0 # define 'letter_count' to 0
    unigram_prob = [] # set 'unigram_prob' as list
    for values in data_occurrence: # loop 'data_occurrence' to keep values of
'data_occurrence' and count number of letters in the data set
        letter_count += data_occurrence[values]
    i = 0 # set 'i' = 0 and is used to point the index of 'data_index'
    for c in data_occurrence: # loop 'data_occurrence' to add values into
'unigram_prob'
        unigram_prob.append((data_index[i], round((data_occurrence[c] / letter_count),
5))) # using append to add value to 'unigram_prob' and calculate unigram probability
        # using unigram formula == (word occurrence) / number of words in data set
        i += 1
    unigram_prob.append(("UNK", round((unk / (letter_count + unk)), 5))) # using
append to add value to 'unigram_prob' and calculate unigram probability of UNK
    print(tabulate(unigram_prob, headers=['Word', 'Probability'], tablefmt='orgtbl'))
#display unigram probability in table

# this function is used to calculate UNIGRAM Laplace smoothing probability of each
word and UNK
def unigram_model_laplace(data_set, unk, vocab, uncover_sentence, is_sentence): #and
also calculate UNIGRAM sentence probability
    #  5 parameters are using in this function 1)data set without <s>,</s> 2)number of
UNK found  3)number of vocabulary 4)sentence data set 5)condition for calculate
```

```
unigram sentence probability
    data_occurrence = collections.Counter(data_set) # 'data_occurrence' is stored
number of occurrences of each word in data set
    data_index = list(data_occurrence) # 'data_index' stores list of 'data_occurrence'
    letter_count = 0 # define 'letter_count' to 0
    unigram_laplace_prob = [] # set 'unigram_laplace_prob' as list
    for values in data_occurrence:   # loop 'data_occurrence' to keep values of
'data_occurrence' and count number of letters in the data set
        letter_count += data_occurrence[values]
    i = 0 # set 'i' = 0 and is used to point the index of 'data_index'
    if is_sentence != "true": # do if this function is called with 'is_sentence' not
equal to "true"
        for c in data_occurrence: # loop 'data_occurrence' to add values into
'unigram_laplace_prob'
            unigram_laplace_prob.append((data_index[i], round((data_occurrence[c] + 1)
/ (vocab + letter_count + 1), 5))) # UNIGRAM LAPLACE smoothing formula is used here
            #(word ocurrence +1) / (number of vocabulary + number of words +1 )
            i += 1
    if is_sentence != "true": # do if this function is called with 'is_sentence' not
equal to "true"
        unigram_laplace_prob.append(("UNK", round(((unk + 1) / (vocab + letter_count +
unk + 1)), 5))) # using append to add value to 'unigram_laplace_prob' and calculate
unigram probability of UNK
        print(tabulate(unigram_laplace_prob, headers=['Word', 'Probability'],
tablefmt='orgtbl')) #display unigram smoothing probability in table

    sentence_prob = [[]] * i # define 'sentence_prob' as list of list * i to store
values of each word probability
    i = 0
    if is_sentence == "true":  # do if this function is called with 'is_sentence'
equal to "true" (calculate sentence probability)
        unk -= 1
        for c in data_occurrence:
            sentence_prob.append([data_index[i], round((data_occurrence[c] + 1) /
(vocab + letter_count + 1), 5)]) # used the same UNIGRAM LAPLACE calculation above to
store each word probability in 'sentence_prob'
            i += 1
        one_d_array = [item for sublist in sampledata_vocab for item in sublist] #
turn sampledata_vocab in to 1D array as 'one_d_array'
        vocab_set = set(one_d_array) # set 'vocab_set' as set of values in
'one_d_array'
        sentence_set = set(uncover_sentence) # set 'sentence_set' as set of values in
'uncover_sentence' (sentence data set)
        sentence_unk = sentence_set.intersection(vocab_set) # intersection each of
above set to get UNK in sentence data set
        my_list = list(sentence_unk) # define 'my_list' as list of 'sentence_unk'
        j = len(sentence_prob) # define 'j' as number of length of sentence_prob
        unigram_model_laplace.num = 0 # 'unigram_model_laplace.num' = 0 (this
parameter will be used to keep value of each sentence probability)
        sum = 0 # set sum = 0 to summation each of the word probability in a sentence
        count = 0 # set count = 0 to count number of loop and will use to check
condition to pass value to 'sum'
        sentence_unk = (len(sentence_set)) - (len(sentence_unk)) # check if there is
UNK in a sentence
        for i in my_list: # outer loop of 'my_list'
            for h in sentence_prob: #inner loop of item in 'sentence_prob'
                if (sentence_prob[j - 1][0]) == i:
                    sum = sentence_prob[j - 1][1] # add every value of word in
sentence to 'sum'
                    count += 1 # now plus 1 to 'count'
                j -= 1 # decrease value of 'j'
                if count == 1:
                    unigram_model_laplace.num = sum # pass valuse of sum to
```

```
unigram_model_laplace.num
                if count > 1 and sentence_prob[j][0] == i:
                    unigram_model_laplace.num = round(unigram_model_laplace.num * sum,
5) # after 1st loop 'unigram_model_laplace.num' will start to calculate sentence
probability by multiply every word probability together
            j = len(sentence_prob) # reset 'j'
        if sentence_unk >= 1: # if there is UNK in a sentence
            unigram_model_laplace.num = round(
                unigram_model_laplace.num * (((unk + 1) / (vocab + letter_count + unk
+ 1)) ** sentence_unk), 5) # unk probability will be add and calculate in to the
sentence probability

# this function will check number of UNK in data set compare with vocab
def unk_count(vocab_lenght, data_uncover): # 2 parameters are used in this 1)vocab 2)
dataset with no <s>, </s>
    unk_count.list = []  #  unk_count.list as list
    for a in sampledata_vocab: # loop to add every vocab into 'unk_count.list'
        if vocab_lenght >= 1:
            unk_count.list += a # adding vocab to list
    set_sample_data = set(data_uncover) # set 'set_sample_data' as a set of
'data_uncover'
    set_vocab_sample_data = set(unk_count.list) # set 'set_vocab_sample_data' as a set
of 'unk_count.list'
    unk_count.unk = len(
        (set_sample_data.union(set_vocab_sample_data) -
set_sample_data.intersection(set_vocab_sample_data))) # now check if we have UNK by
union then - with data intersection vocab

# this function creates bigram pairs of data set
def into_bigram_pair(data):  # 1 parameter is used here is 'data' (original data set
with <s>, </s>)
    data = [item for sublist in data for item in sublist] # loop to make item in data
set into one list
    text = " ".join(data) # text = join every item in data using space to split
    into_bigram_pair.data = list(nltk.bigrams(text.split())) # this called nltk
package to split words into pairs (bigram)
    index = len(into_bigram_pair.data) - 1 # set 'index' to length of
into_bigram_pair.data to use for looping
    for item in into_bigram_pair.data: # loop to remove the pair that contains </s> as
the first word in the pair
        if into_bigram_pair.data[index][0] == "</s>":
            del into_bigram_pair.data[index] # del that pair 'into_bigram_pair.data'
at 'index' count
        index -= 1 # decrease index evertime of the loop

# this fucntion is use to calculate the probability of BIGRAM unsmoothing using
dataset without cutting starting and ending sentence
def bigram_unsmoothed(data, bigram_data, unmatched_set): #and also uses bigram pairs
of data set, and 'unmatched set' means the posible pairs that not occur in the bigramd
data
    unmatched_set = collections.Counter(unmatched_set) #count the number of unmatched
pair
    bigram_unsmoothed.ind = collections.Counter(data) # count the number of each word
occured in the data set given
    bigram_unsmoothed.pair = collections.Counter(bigram_data) # count the number of
pairs in the data set given
    bigram_list = list(bigram_unsmoothed.ind) # 1
    pair_values = list(bigram_unsmoothed.pair) # 2
    unknown_pair = list(unmatched_set) # 3 # from 1-3 turn the counting information
above into lists
    i = 0
    j = len(bigram_list) # count the length of data set counting
    bigram_unsmoothed_prob = [] # define 'bigram_unsmoothed_prob' as a list
```

```
    for c in bigram_unsmoothed.pair: # outer loop = 'bigram_unsmoothed.pair' items
        for x in range(j): # inner loop = 'length of data set count'
            if i == len(pair_values):  # break before number of index in
'pair_values' exceeding the last index
                break
            if bigram_list[j - 1] == pair_values[i][0]: # do if finds word match with
the first pair word
                x = (list(bigram_unsmoothed.ind.values())[j - 1]) # pass values of the
number of pair into 'x'
                bigram_unsmoothed_prob.append((pair_values[i],
round((bigram_unsmoothed.pair[c]) / (x), 5))) # use the bigram formula to calculate
the prob. of each pair and add the value into 'bigram_unsmoothed_prob'
                ## along with the particular pair data
                i += 1
            j -= 1
        j = len(bigram_list) # reset the length of j to loop again until the outer
loop ends
    j = len(bigram_list) # again set j equal to length of counted data set
    i = 0
    for c in unmatched_set: # this loop will do the same way of the above loop but use
the not occur pair in the bigram of data set
        for x in range(len(bigram_list)):
            if i == len(unknown_pair):
                break
            elif bigram_list[j - 1] == unknown_pair[i][0]:
                bigram_unsmoothed_prob.append((unknown_pair[i], 0)) # we don't need
formula here because this always 0 (not like add-one smoothing that can have values)
                i += 1
            j -= 1
        j = len(bigram_list)
    i = 0
    j = len(bigram_list)
    for c in bigram_unsmoothed.pair: # this loop used to calculate the prob. of pair
that has UNK or UNK and UNK in the pair
        if i == j:
            break # use break before counting of 'i' exceed the last index in the list
in 'bigram_list'
        num = (list(bigram_unsmoothed.ind.values())[i]) # store value of counted pair
in data set
        bigram_unsmoothed_prob.append(((bigram_list[i], 'UNK'), round(unk_count.unk /
(num + len(bigram_list)), 5))) # now add the values of calculation into
'bigram_unsmoothed_prob' list to use later
        bigram_unsmoothed_prob.append((("UNK", bigram_list[i]), round(unk_count.unk /
len(bigram_list), 5)))  # now add the values of calculation into
'bigram_unsmoothed_prob' list to use later
        i += 1
    bigram_unsmoothed_prob.append((("UNK", "UNK"), unk_count.unk / len(bigram_list)))
# calculate prob. of UNK and UNK in the same pair and add into
'bigram_unsmoothed_prob'
    print(tabulate(bigram_unsmoothed_prob, headers=['Pairs occurrence',
'Probability'], tablefmt='orgtbl')) # display all of the prob. in table


# this function does the same method of function 'bigram_unsmoothed' but use the
LAPLACE smoothing formula to calculate the prob. of bigram
def bigram_smoothed_laplace(data, bigram_data, unmatched_set): # use the same
parameters as 'bigram_unsmoothed' uses
    unknown_pair = list(unmatched_set)
    unk_count.unk += 1
    bigram_smoothed_laplace.ind = collections.Counter(data)
    list_data_count = list(bigram_smoothed_laplace.ind)
    bigram_smoothed_laplace.pair = collections.Counter(bigram_data)
    arr = list(bigram_smoothed_laplace.pair)
```

```
    i = 0
    j = len(list_data_count)
    bigram_smoothed_laplace.store_prob = []
    bigram_smoothed_laplace_prob = []
    for C in bigram_smoothed_laplace.pair:
        for x in range(len(list_data_count)):
            if i == len(arr):
                break
            if list_data_count[j - 1] == arr[i][0]:
                counted_data_letter = list(bigram_smoothed_laplace.ind.values())[j -
1]
                counted_pair = bigram_smoothed_laplace.pair[C]
                bigram_smoothed_laplace.store_prob.append([arr[i], ((((counted_pair +
1) / (
                        counted_data_letter + len(list_data_count))))])
                bigram_smoothed_laplace_prob.append((arr[i], round(((counted_pair + 1)
/ (
                        counted_data_letter + len(list_data_count))), 5)))
                i += 1
            j -= 1
        j = len(list_data_count)

    i = 0
    k = 2
    for x in unknown_pair:
        if i == len(unknown_pair) or k < 0:
            break
        counted_data_letter = list(bigram_smoothed_laplace.ind.values())[k]
        bigram_smoothed_laplace.store_prob.append([unknown_pair[i], ((1 /
((counted_data_letter) + len(list_data_count))))])
        bigram_smoothed_laplace_prob.append((unknown_pair[i], round(1 /
((counted_data_letter) + len(list_data_count)), 5)))
        k -= 1
        i += 1
    i = 0
    for c in bigram_smoothed_laplace.pair:
        if i == j:
            break
        counted_data_letter = list(bigram_smoothed_laplace.ind.values())[i]
        bigram_smoothed_laplace.store_prob.append(
            [(list_data_count[i], 'UNK'), (unk_count.unk / (counted_data_letter +
len(list_data_count)))])
        bigram_smoothed_laplace_prob.append(((list_data_count[i], 'UNK '),
round((unk_count.unk / (counted_data_letter + len(list_data_count))), 5)))
        bigram_smoothed_laplace.store_prob.append([("UNK", list_data_count[i]),
(unk_count.unk / len(list_data_count))])
        bigram_smoothed_laplace_prob.append((("UNK", list_data_count[i]),
round(unk_count.unk / len(list_data_count), 5)))
        i += 1
    bigram_smoothed_laplace.store_prob.append([("UNK", "UNK"), (unk_count.unk /
len(list_data_count))])
    bigram_smoothed_laplace_prob.append((("UNK", "UNK"), round(unk_count.unk /
len(list_data_count), 5)))
    print(tabulate(bigram_smoothed_laplace_prob, headers=['Pairs occurrence',
'Probability'], tablefmt='orgtbl'))

# this function calculates sentence probability by multiplying each of bigram data in
each sentence using values of bigram laplace smoothing we have stored
def bigram_sentence_probabality(data, text_bigram):
    bigram_sentence_probabality.prob = 0
    i = 0
    count = 0
    num = 0
```

```
    j = len(text_bigram)
    for c in data:
        for x in text_bigram:
            if text_bigram[j - 1][0] == data[i]:
                num = text_bigram[j - 1][1] # pass the value of 'text_bigram' to 'num'
                count += 1
            j -= 1
            if count == 1:
                bigram_sentence_probabality.prob = num # pass the value of num to
'bigram_sentence_probabality.prob'
            if count > 1 and text_bigram[j][0] == data[i]:
                bigram_sentence_probabality.prob = bigram_sentence_probabality.prob *
num # now multiply them together
        i += 1
        j = len(text_bigram)

# this function calculates all of the possible pair in the BIGRAM
def all_possible_pair():
    all_possible_pair.store_all_pos = []
    flat = [item for sublist in sampledata_vocab for item in sublist] # makes the list
of vocab list into just list of vocab
    flat.insert(0, "<s>") # add <s> at first index
    flat.append("</s>") # add </s> at the last index
    for c in flat: # loop at number of item in 'flat' (new vocab with <s>,</s>) by
'flat'
        for x in flat:
            all_possible_pair.store_all_pos.append((c, x)) # add all of the possible
pair into 'store_all_pos'
    all_possible_pair.store_all_pos = list(all_possible_pair.store_all_pos)
    i = len(all_possible_pair.store_all_pos) - 1
    for a in all_possible_pair.store_all_pos: # now loop agian to remove </s> that
occurs in the first word of pair
        if all_possible_pair.store_all_pos[i][0] == "</s>":
            del all_possible_pair.store_all_pos[i] # this is how I remove that item
            i = i - 1
    i = len(all_possible_pair.store_all_pos) - 1

    unwanted_pair = [('<s>', '<s>'), ('<s>', '</s>')] # set the pairs that we don't
want in the bigram data
    y = 0
    for i in range(26): # loop 26 time (from a-z) to add the pair that we don't to
calculate into 'unwanted_pair' means that a-z followed by starting sentence
        unwanted_pair.append((chr(ord('a') + y), '<s>'))
        y += 1
    set_unwanted = set(unwanted_pair) #1
    prm_first_set = set(all_possible_pair.store_all_pos) #2
    prm_second_set = set(into_bigram_pair.data) #4
    unmatched_set = prm_first_set - prm_second_set #5
    all_possible_pair.unmatched_set = unmatched_set - set_unwanted # now I extract and
get only the pair that we don't have yet in the bigram data by substracting each set
    all_possible_pair.sampledata_cover = [item for sublist in read_sampledata_cover
for item in sublist] # turn the 'read_sampledata_cover' (data set with <s>, </s>) into
sampledata_cover as list

# this function will check the number of UNK in the sentence test given and change
every word in the sentence that is UNK to UNK
def sort_list(sentence_data): # 1 parameter needs is sentence data
    sentence = [item for sublist in sentence_data for item in sublist] #1
    store_raw_vocab = [item for sublist in all_possible_pair.store_all_pos for item in
sublist] #2
    set_sentence = [item for sublist in sentence_data for item in sublist] #3
    # from 1-3 I changed every list of list into list to turn them into 'set'
    set_sentence = set(set_sentence) # turn list of sentence into set
```

```
    set_vocab = set(store_raw_vocab)  # turn list of vocab into set
    intersec_bigram_and_vocab = set_sentence - set_vocab # now check the UNK in the
sentence by substract the sentence with the set of vocab

    arr_sentence = []
    arr_intersec_bigram_and_vocab = []
    for s in sentence: # loop to add word in the sentence in to 'arr_sentence'
        arr_sentence.append([s])
    for s in intersec_bigram_and_vocab: # loop to add the words that are UNK in
'arr_intersec_bigram_and_vocab'
        arr_intersec_bigram_and_vocab.append([s])

    k = 0
    for i in arr_sentence:
        for j in arr_intersec_bigram_and_vocab:
            if i == j:
                del arr_sentence[k] # loop to delete that UNK word in the sentence
                arr_sentence.insert(k, ["UNK"]) # and insert "UNK" instead
        k = k + 1

    sort_list.new_sentence = [[]] # define list of list
    for s in arr_sentence: # now turn into list of list again with new value
        sort_list.new_sentence.append([s])

####### read the files by calling 'read_file' function and store the values in
parameter as list######
read_sampledata = read_file('sampledata.txt') #
read_sampledata_cover = read_file('sampledata.txt')
sampledata_vocab = read_file('sampledata.vocab.txt')
sentence_data = read_file('sampletest.txt')
####### read the files by calling 'read_file' function and store the values in
parameter as list######


vocab_lenght = len(sampledata_vocab) # 'vocab_lenght' as the 'sampledata_vocab' length
remove_cover(read_sampledata, "model") # call function 'remove_cover' to remove <s>,
</s> of sample data
unk_count(vocab_lenght, remove_cover.data) # count the number of UNK by using function
'unk_count'
print("Training data (sampledata.txt) : ", read_sampledata_cover) #print the read
files of 'read_sampledata_cover'
print("Vocabulary (sampledata.vocab.txt) : ", sampledata_vocab) #print the read files
of 'sampledata_vocab'
print("UNK Found : ", unk_count.unk) # display the number of UNK
print("\nxxxxxxxxxxxxxxx U N I G R A M xxxxxxxxxxxxxxx (UNSMOOTHED)")
unigram_model(remove_cover.data, unk_count.unk) #calculate the UNIGRAM prob. of word
using 'unigram_model' function
print("xxxxxxxxxxxxxxx U N I G R A M xxxxxxxxxxxxxxx (SMOOTHED)")
unigram_model_laplace(remove_cover.data, unk_count.unk, len(unk_count.list), 0,
"false")  # calculate the UNIGRAM LAPLCE SMOOTHING prob. of word using
'unigram_model_laplace' function

print("\nxxxxxxxxxxxxxxxxx B I G R A M xxxxxxxxxxxxxxxxx (UNSMOOTHED)")
into_bigram_pair(read_sampledata_cover) # send the sample data without removing <s>,
</s> to get pairs of BIGRAM MODEL
all_possible_pair() # now use the bigram data from above line to calculate all
possible pairs in the bigram prob. table
bigram_unsmoothed(all_possible_pair.sampledata_cover, into_bigram_pair.data,
all_possible_pair.unmatched_set)
# send all the information we got through the 'bigram_unsmoothed' function to
calculate bigram unsmoothing prob.

print("\nxxxxxxxxxxxxxxxxx B I G R A M xxxxxxxxxxxxxxxxx (SMOOTHED)")
```

```
bigram_smoothed_laplace(all_possible_pair.sampledata_cover, into_bigram_pair.data,
all_possible_pair.unmatched_set)
# call 'bigram_smoothed_laplace' function to calculate bigram laplace smoothing using
the same information as unsmoothing used

print("\nxxxxxxxxxxxxxxx S E N T E N C E _ P R O B A B I L I T Y xxxxxxxxxxxxxxxx")
print("Sentence Data(sampletest.txt) : ", sentence_data) #display the every sentences
data in the sampletest.txt
remove_cover(sentence_data, "prob") #call 'remove_cover' to remove <s>, </s> from
sentence data
sentence = read_file('sampletest.txt')#read the files by calling 'read_file' function
and store the values in 'sentence' as list
unigram_model_laplace(remove_cover.data, unk_count.unk, vocab_lenght,
sentence_data[2], "true") #calculate the unigram laplace smoothing of sentence 3 using
'unigram_model_laplace'
print("Sentence no.3",sentence[2], "     : Unigram Prob. :",
unigram_model_laplace.num) # print the result of sentence 3
unigram_model_laplace(remove_cover.data, unk_count.unk, vocab_lenght,
sentence_data[3], "true") #calculate the unigram laplace smoothing of sentence 4 using
'unigram_model_laplace'
print("Sentence no.4",sentence[3], ": Unigram Prob. :", unigram_model_laplace.num) #
print the result of sentence 4
unigram_model_laplace(remove_cover.data, unk_count.unk, vocab_lenght,
sentence_data[4], "true") #calculate the unigram laplace smoothing of sentence 5 using
'unigram_model_laplace'
print("Sentence no.5",sentence[4], ": Unigram Prob. :", unigram_model_laplace.num) #
print the result of sentence 5

sort_list(read_file('sampletest.txt')) # call the 'sort_list' function to get new
sentence information to use in another function
new_sentence = [item for sublist in sort_list.new_sentence for item in sublist] # turn
the information from 'sort_list.new_sentence' into list named 'new_sentence'
into_bigram_pair(new_sentence) #now send the 'new_sentence' through 'into_bigram_pair'
function
bigram_sentence_probabality(into_bigram_pair.data[10:14],
bigram_smoothed_laplace.store_prob) # calculate bigram laplace smoothing prob. of
sentence 3(array index) using function 'bigram_sentence_probabality'
print("Sentence no.3",sentence[2], "     : Bigram Prob. :",
round((bigram_sentence_probabality.prob), 5)) #display the result of sentence 3 prob.
bigram_sentence_probabality(into_bigram_pair.data[14:19],
bigram_smoothed_laplace.store_prob) # calculate bigram laplace smoothing prob. of
sentence 4(array index) using function 'bigram_sentence_probabality'
print("Sentence no.4",sentence[3], ": Bigram Prob. :",
round((bigram_sentence_probabality.prob), 5)) #display the result of sentence 3 prob.
bigram_sentence_probabality(into_bigram_pair.data[19:24],
bigram_smoothed_laplace.store_prob) # calculate bigram laplace smoothing prob. of
sentence 5(array index) using function 'bigram_sentence_probabality'
print("Sentence no.5",sentence[4], ": Bigram Prob. :",
round((bigram_sentence_probabality.prob), 5)) #display the result of sentence 3 prob.
```

Answer.5 a and b

```
xxxxxxxxxxxxxx U N I G R A M xxxxxxxxxxxxxxx (UNSMOOTHED)
| Word    |   Probability |
|---------+---------------|
| a       |       0.26667 |
| b       |       0.33333 |
| c       |       0.4     |
| UNK     |       0       |
xxxxxxxxxxxxxx U N I G R A M xxxxxxxxxxxxxxx (SMOOTHED)
| Word    |   Probability |
|---------+---------------|
| a       |       0.26316 |
| b       |       0.31579 |
| c       |       0.36842 |
| UNK     |       0.05263 |
```

Q.6 (20%) Computing a bigram model Use the Toy dataset. The vocabulary is the words in sampledata.vocab.txt, plus the UNK token and symbols. should be included only in the context or history. should not be included in the history but only as the following word. The table below should clarify for you. a) Compute the probabilities in a bigram language model without smoothing. Show your work for P(b|a), P(UNK|), P(UNK|UNK). Create a table in the following format and list all of the probabilities in the model.

b) Smooth the model using Laplace smoothing. Show your work for P(b|a), P(UNK|), P(UNK|UNK). Show all the smoothed probabilities in a table.

Answer. 6 a and b

xxxxxxxxxxxxxxxxx B I G R A M xxxxxxxxxxxxxxxxxxx (UNSMOOTHED)

Pairs occurrence	Probability
('<s>', 'a')	0.66667
('a', 'a')	0.25
('a', 'b')	0.5
('b', 'b')	0.2
('b', 'c')	0.6
('c', 'c')	0.33333
('c', '</s>')	0.33333
('a', 'c')	0.5
('c', 'b')	0.16667
('<s>', 'b')	0.33333
('c', 'a')	0.16667
('b', '</s>')	0.2
('<s>', 'c')	0
('a', '</s>')	0
('b', 'a')	0
('<s>', 'UNK')	0
('UNK', '<s>')	0
('a', 'UNK')	0
('UNK', 'a')	0
('b', 'UNK')	0
('UNK', 'b')	0
('c', 'UNK')	0
('UNK', 'c')	0
('</s>', 'UNK')	0
('UNK', '</s>')	0
('UNK', 'UNK')	0

xxxxxxxxxxxxxxxxx B I G R A M xxxxxxxxxxxxxxxxxxx (SMOOTHED)

Pairs occurrence	Probability
('<s>', 'a')	0.375
('a', 'a')	0.22222
('a', 'b')	0.33333
('b', 'b')	0.2
('b', 'c')	0.4
('c', 'c')	0.27273
('c', '</s>')	0.27273
('a', 'c')	0.33333
('c', 'b')	0.18182
('<s>', 'b')	0.25
('c', 'a')	0.18182
('b', '</s>')	0.2
('<s>', 'c')	0.1
('a', '</s>')	0.11111
('b', 'a')	0.125
('<s>', 'UNK ')	0.125
('UNK', '<s>')	0.2
('a', 'UNK ')	0.11111
('UNK', 'a')	0.2
('b', 'UNK ')	0.1
('UNK', 'b')	0.2
('c', 'UNK ')	0.09091
('UNK', 'c')	0.2
('</s>', 'UNK ')	0.125
('UNK', '</s>')	0.2
('UNK', 'UNK')	0.2

Q.7 (10%) Computing sentence probabilities Use the Toy dataset. There are 5 sentences in sampletest.txt. Using the smoothed models above, compute the probability of each sentence. For unigram probability, you should ignore the and symbols.

a) Show your work for sentence numbers 3, 4, 5, for each model: unigram and bigram.

b) Fill in the probabilities of all the sentences in a table.

Answer. 7 a and b

```
xxxxxxxxxxxxxxx S E N T E N C E _ P R O B A B I L I T Y xxxxxxxxxxxxxxxxx
Sentence no.3 ['<s>', 'c', 'b', 'a', '</s>']      : Unigram Prob. : 0.03062
Sentence no.4 ['<s>', 'a', 'b', 'c', 'd', '</s>'] : Unigram Prob. : 0.00161
Sentence no.5 ['<s>', 'a', 'd', 'e', 'b', '</s>'] : Unigram Prob. : 0.00023
Sentence no.3 ['<s>', 'c', 'b', 'a', '</s>']      : Bigram Prob. : 0.00025
Sentence no.4 ['<s>', 'a', 'b', 'c', 'd', '</s>'] : Bigram Prob. : 0.00091
Sentence no.5 ['<s>', 'a', 'd', 'e', 'b', '</s>'] : Bigram Prob. : 0.00033
```