/***

Name: Aadish Joshi
Email: asj170430@utdallas.edu
Number of problems solved: 4

***/

Problem 1:

Q: Write regular expressions for the following three languages. By "word", we mean an alphabetic string separated from other words by whitespace, any relevant punctuation, line breaks, and so forth.
1. Language 1: the set of all strings with two consecutive repeated words (e.g., "Humbert Humbert" and "the the" but not "the bug" or "the big bug"); (1 point)
2. Language 2: all strings that start at the beginning of the line with an integer and that end at the end of the line with a word; (2 points)
3. Language 3: all strings that have both the word "grotto" and the word "raven" in them (but not, e.g., words like grottos that merely contain the word grotto); (2 points)

Regex 1: \b(\w+)\s+\1\b
Explanation: \w+ means one more more occurrence of word and numeric. Also \1 signifies repeating by 1 so that the bug and the big bug will not be selected.

Regex 2: ^[\d].*[a-zA-Z]$
Explanation:
We want string to start with an integer and end with a word. \w cannot be used at the end as it will characterize words and numbers. So we specifically use **[a-zA-Z]** at the end. **^** and **$** signify start and end of the sentence. **.*** is 0 or more occurrence of anything in between.

Regex 3: .*(\bgrotto\b.*\braven\b|\braven\b.*\bgrotto\b).*
Explanation:
| signify OR statement. Hence anything with confined boundary \b grotto \b and \braven\b or anything between raven and grotto will be selected.

Problem 2:

Question:
1. Write a program to compute the bigrams for any given input. ***TOTAL: 5 points***
Apply your program to compute the bigrams you need for sentences S1 and S2.
2. Construct automatically (by the program) the tables with (a) the bigram counts (5 points) and the (b) bigram probabilities for the language model without smoothing. (5 points) ***TOTAL: 10 points***
3. Construct automatically (by the program): (i) the Laplace-smoothed count tables; (5 points) (ii) the Laplace-smoothed probability tables (5 points); and (iii) the corresponding re-constituted counts (5 points) ***TOTAL: 15 points***
4. Compute the total probabilities for each sentence S1 and S2, when (a) using the bigram model without smoothing; (5 points) and (b) when using the bigram model Laplace-smoothed. (5 points) ***TOTAL: 10 points***

Total Probability for the sentence s1:
`Without smoothing:` 3.623421529813562e-13
`With add 1 smoothing:` 1.8887153744687491e-28

Total Probability of the sentence s2:
`Without smoothing:` 1.0309230980059047e-13
`With add 1 smoothing:` 3.26520268778309 96e-33

Programming Language: Python3

Software tool used: Jupyter Notebook (Anaconda Navigator)
Primary data structure used: Dictionary

Supporting file:
Create following files (Already included in submitted code)
1) "inputforbigrams.txt" containing corpus of the problem 2
2) "testbigramInput.txt" containing lines for the test input.

Programming imports:

```
In [1]: import nltk
        import random
        import re
        import numpy as np
```

Code description:

1) Firstly, the data is read through "inputforbigrams.txt" file for training corpus model.

```
In [5]: #readfile
        myfile = open("inputforbigrams.txt","r")

        paragraph = myfile.readlines()

        data = preprocess(paragraph)
```

2) It is pre-processed by adding <s> and </s>syntaxes at the start and the end of every sentence in the paragraph. No removal of punctuations as they are considered as a individual token in bigrams.

In [4]:
```
def preprocess(paragraph):

    data = ""

    for sentences in paragraph:
        data += '<s> '+sentences.strip()+' </s> ';

    return data
```

3) Individual count for unigram and then bigrams is calculated

In [9]:
```
def unigrams(data):
    unigramSplit = data.split()
    return unigramSplit
```

In [10]:
```
def bigrams(data):
    unigramSplit = data.split()
    return list(zip(unigramSplit, unigramSplit[1:]))
```

4) Individual probability is calculated as per (Bigram count / Unigram count)

```
probabilities = {}

for i in unigramCounts:
    for j in unigramCounts:
        try:
            Ci = bigramCountsDict[(i[0], j[0])]
            N  = unigramCountsDict[i[0]];
            probabilities[(i[0], j[0])] = Ci/N
        except:
            probabilities[(i[0], j[0])] = 0

#print(probabilities)
```

5) Bigram count is calculated by adding Laplace smoothing as well by adding 1 to individual token count.

```
LaplaceianProbs = {}

for i in unigramCounts:
    for j in unigramCounts:


        N = unigramCountsDict[i[0]]
        V = Vocabulary

        try:
            Ci = bigramCountsDict[(i[0], j[0])]
            LaplaceianProbs[(i[0], j[0])] = (1+Ci)/(V+N)
        except:
            LaplaceianProbs[(i[0], j[0])] = (1)/(V+N)
#print(LaplaceianProbs)
```

```
reconProbs = {}

for i in unigramCounts:
    for j in unigramCounts:
        N = unigramCountsDict[i[0]]
        Ci_1 = N
        V = Vocabulary
        try:
            Ci = bigramCountsDict[(i[0], j[0])]
            reconProbs[(i[0], j[0])] = (1 + Ci)*(Ci_1)/(V+N)
        except:
            reconProbs[(i[0], j[0])] = Ci_1/(V + N)
print(reconProbs)
```

5) Testing data is again split in bigrams.

```
f = open("testbigramInput.txt","r")

test_paragraph = f.readlines()

testdata = preprocess(test_paragraph)
```

```
bigramTestSplit = bigrams(testdata)
print(bigramTestSplit)
```

```
[('<s>', 'Sales'), ('Sales', 'of'), ('of', 'the'), ('the', 'company'), ('company', 'to'), ('to', 'return'), ('return', 'to'), ('to', 'normalcy'), ('normalcy', '.'), ('.', '</s>')]
```

6) Referring the dictionary probability is calculated for each sentence.

```
TestProb = 1;
for i in bigramTestSplit:
    try:
        print(probabilities[i])
        TestProb = TestProb * probabilities[i]
    except:
        print("notfound")

print(TestProb)
```

```
0.001
1.0
0.2835820895522388
0.05874125874125874
0.013605442176870748
0.0030816640986132513
0.36363636363636365
0.0015408320493066256
1.0
0.9259259259259259
3.623421529813562e-13
```

```
TestLapProb = 1;
for i in bigramTestSplit:
    try:
        print(LaplaceianProbs[i])
        TestLapProb = TestLapProb * LaplaceianProbs[i]
    except:
        print("notfound")

print(TestLapProb)
```

```
0.0003026634382566586
0.00035656979853806385
0.04078826764436297
0.012077294685990338
0.0005212858384013901
0.000479463001438389
0.0008898380494749955
0.000319642000958926
0.00035656979853806385
0.14015438171636144
1.8887153744687491e-28
```

Problem 3:

Considering the same corpus as in Problem 2, write a program to compute the Positive Pointwise Mutual Information (PPMI) of the following words and present the results in a term-context matrix. The context of a context-word is the "window" of words consisting of (i) 5 words to the left of the context-word; (ii) the context-word; and (iii) 5 words to the right of the context-word. IF there are fewer then 5 words to the right or the left of the context-word in the same sentence, the context will be padded with "NIL". Compute the PPMI and populate the term-context matrix for:

a. The word "chairman" for the context-word "said";

b. The word "chairman" in the context-word "of";

c. The word "company" in the context-word "board";

d. The word "company" in the context-word "said".

TOTAL: 10 points

2. Use add-2 smoothing for the same words and contexts as in (1) and present the result in a term-context matrix. ***TOTAL: 5 points***

3. Find which words are more similar among: [chairman, company], [company, sales] or [company, economy] when considering only the contexts provided by the context-words "said", "of", and "board"? Explain why. ***TOTAL: 5 points***

4. Using the pre-trained Glove embeddings available at: https://nlp.stanford.edu/projects/glove/ which words are more similar among: [chairman, company], [company, sales] or [company, economy]? Explain why. ***TOTAL: 5 points***

1) PPMI

```
[Chairman, said] = 0.06938
[Chairman, of] = 0.06130258
[company, board] = 1.678362
[company, said] = 0.0
```

```
def ppmi_matrix(PMIMatrix):
    (row,col) = PMIMatrix.shape

    PPMIMatrix = np.zeros((row,col))

    for i in range(row):
        for j in range(col):
            pmi = PMIMatrix[i,j]
            if(pmi < 0):
                PPMIMatrix[i,j] = 0
            else:
                PPMIMatrix[i,j] = pmi
    return PPMIMatrix
print(ppmi_matrix(PMIMatrix))
```

```
[[0.06938949 0.06130258 0.        ]
 [0.         0.         1.67836234]]
```

2)
add-2 smoothing

Result:

```
[Chairman, said] = 0.12485041
[Chairman, of] = 0.1031523
[company, board] = 1.81727949
[company, said] = 0.0
```

```
def ppmi_matrix_smoothed(PMIMatrix_smoothed):
    (row,col) = PMIMatrix_smoothed.shape

    PPMIMatrix_smoothed = np.zeros((row,col))

    for i in range(row):
        for j in range(col):
            pmi = PMIMatrix_smoothed[i,j]
            if(pmi < 0):
                PPMIMatrix_smoothed[i,j] = 0
            else:
                PPMIMatrix_smoothed[i,j] = pmi
    return PPMIMatrix_smoothed
print(ppmi_matrix_smoothed(PMIMatrix_smoothed))
```

```
[[0.12485041 0.1031523  0.        ]
 [0.         0.         1.81727949]]
```

3)
Results:
Chairman, company : 0
Company, sales: 0.3473
Company, economy: 0.4841

Explanation:
We see that company economy appears to be more similar. The reason for their similarity is two words occurred approximately equal number of times in terms of context words. Hence their word vectors appear to be close as word vectors are calculated based on count of occurrence.

```
#************* Testing PROBLEM 3 *********
```

```
chairman_said = find_term_context_count(paragraph, "chairman", "said")
chairman_of = find_term_context_count(paragraph, "chairman", "of" )
chairman_board = find_term_context_count(paragraph, "chairman", "board" )

company_said = find_term_context_count(paragraph, "company", "said")
company_of = find_term_context_count(paragraph, "company", "of")
company_board = find_term_context_count(paragraph, "company", "board")

sales_said = find_term_context_count(paragraph, "sales", "said")
sales_of = find_term_context_count(paragraph, "sales", "of")
sales_board = find_term_context_count(paragraph, "sales", "board")

economy_said = find_term_context_count(paragraph, "economy", "said")
economy_of = find_term_context_count(paragraph, "economy", "of")
economy_board = find_term_context_count(paragraph, "economy", "board")
```

```
TCM = np.zeros((4,3))

TCM[0,0] = chairman_said
TCM[0,1] = chairman_of
TCM[0,2] = chairman_board

TCM[1,0] = company_said
TCM[1,1] = company_of
TCM[1,2] = company_board

TCM[2,0] = sales_said
TCM[2,1] = sales_of
TCM[2,2] = sales_board

TCM[3,0] = economy_said
TCM[3,1] = economy_of
TCM[3,2] = economy_board

print(TCM)
```

```
[[122. 292.  15.]
 [ 23.  57.  26.]
 [  3.   4.   0.]
 [  1.   4.   0.]]
```

```
N = np.sum(TCM)
p_all_context = np.sum(TCM, axis=0)
print(p_all_context)
p_all_information = np.sum(TCM, axis=1)
print(p_all_information)
```

```
[149. 357.  41.]
[429. 106.   7.   5.]
```

```
import math
def pmi_matrix(TCM, N):

    (row,col) = TCM.shape

    PMI = np.zeros((row,col))

    for i in range(row):
        for j in range(col):
            pij = TCM[i,j] / N
            pw = p_all_information[i] / N
            pc = p_all_context[j] / N
            try:
                PMI[i,j] = math.log((pij/(pw*pc)),2)
            except:
                PMI[i,j] = 0
    return PMI

PMIMatrix = pmi_matrix(TCM, N)
print(PMIMatrix)
```

```
[[ 0.062132    0.06060748 -1.10009822]
 [-0.32813    -0.27941368  1.71036428]
 [ 0.65383608 -0.19173816  0.        ]
 [-0.44569959  0.29368866  0.        ]]
```

```
def ppmi_matrix(TCM):
    (row,col) = TCM.shape

    PPMI = np.zeros((row,col))

    for i in range(row):
        for j in range(col):
            pmi = TCM[i,j]
            if(pmi < 0):
                PPMI[i,j] = 0
            else:
                PPMI[i,j] = pmi
    return PPMI
print(ppmi_matrix(PMIMatrix))
```

```
[[0.062132   0.06060748 0.        ]
 [0.         0.         1.71036428]
 [0.65383608 0.         0.        ]
 [0.         0.29368866 0.        ]]
```

4)

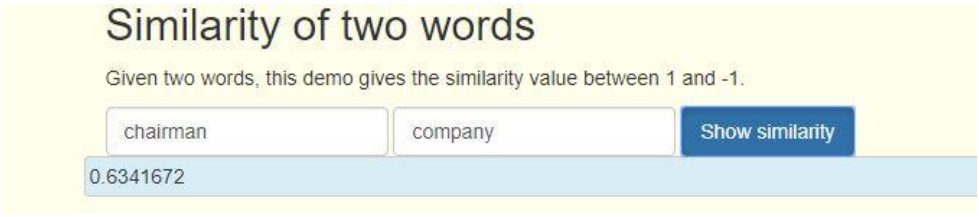
Word similarity using GloVe

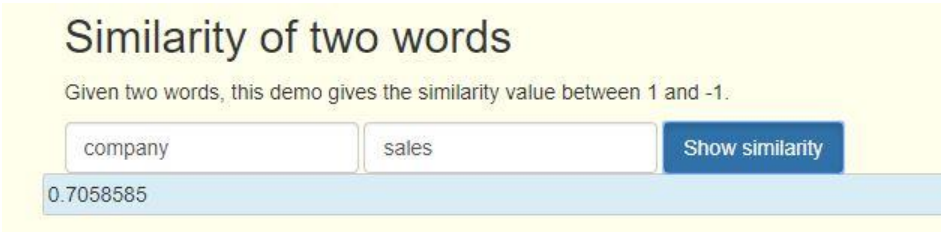
Results:

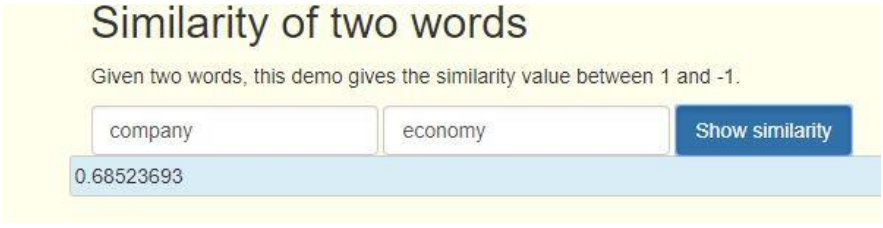
Chairman, company : 0.63416
Company, sales: 0.7058585
Company, economy: 0.6852

Explanation :

Using glove through online website, the similarity of company to sales appear to be more. This similarity is **corpus specific**. Hence in greater terms where in corpus, if sales occurred approximately equal to the company in some corpus, those words will be rated near similar.







Codes:

Programming Language: Python3

Software tool used: Jupyter Notebook (Anaconda Navigator)

Primary data structure used: Dictionary

Supporting file:
Create following files (Already included in submitted code)
1) "inputforbigrams.txt" containing corpus of the problem 2
2) "testbigramInput.txt" containing lines for the test input.

Programming imports:

```
import nltk
import random
import re
import numpy as np
```

Code description:

1) We do not need to pre-process the sentences in the paragraph as we have to find the count of each term word in its context.

2) I have written find_term_context_count which returns count of that term in the window of 5 words before and 5 words after the terms. As the tokens like punctuations do not add value to the term context matrix, I removed to punctuations and other tokens using 'nltk.word_tokenizer'.

```
def find_term_context_count(paragraph, context, term):
    count = 0

    for sentences in paragraph:
        words = nltk.word_tokenize(sentences)
        words=[word.lower() for word in words if word.isalpha()]

        if context in words:
            word_index = words.index(context)

            if term in (words[max(0,word_index-5):min(word_index+6,len(words))]):
                count += 1

    return count
```

3) Then count of word chairman in context of “said”, [chairman, of] , [chairman, board], [company, said], [company, of], [company, board] are calculated with the help of function.

```
chairman_said = find_term_context_count(paragraph, "chairman", "said")
chairman_of = find_term_context_count(paragraph, "chairman", "of" )
chairman_board = find_term_context_count(paragraph, "chairman", "board" )

company_said = find_term_context_count(paragraph, "company", "said")
company_of = find_term_context_count(paragraph, "company", "of")
company_board = find_term_context_count(paragraph, "company", "board")

print(chairman_said)

print(chairman_of)

print(chairman_board)

print(company_said)

print(company_of)

print(company_board)
```

```
89
277
13
20
47
26
```

4) A term context matrix has been created and filled in with the individual values.

```
TermContextMatrix = np.zeros((2,3))

TermContextMatrix[0,0] = chairman_said
TermContextMatrix[0,1] = chairman_of
TermContextMatrix[0,2] = chairman_board

TermContextMatrix[1,0] = company_said
TermContextMatrix[1,1] = company_of
TermContextMatrix[1,2] = company_board

print(TermContextMatrix)
```

```
[[ 89. 277.  13.]
 [ 20.  47.  26.]]
```

5) total N count has been calculated

```
N = np.sum(TermContextMatrix)
print(N)
```

```
535.0
```

```
p_context = np.sum(TermContextMatrix, axis=0)
print(p_context)
```

```
[145. 349.  41.]
```

```
p_information = np.sum(TermContextMatrix, axis=1)
print(p_information)
```

```
[429. 106.]
```

6) PMIMatrix[I,j] = Log_2(Count / N)/(((p(context)/N) * ((P(information)/N))

```
import math
def pmi_matrix(TermContextMatrix, N):

    (row,col) = TermContextMatrix.shape

    PMIMatrix = np.zeros((row,col))

    for i in range(row):
        for j in range(col):
            pij = TermContextMatrix[i,j] / N
            pw = p_information[i] / N
            pc = p_context[j] / N
            try:
                PMIMatrix[i,j] = math.log((pij/(pw*pc)),2)
            except:
                PMIMatrix[i,j] = 0
    return PMIMatrix

PMIMatrix = pmi_matrix(TermContextMatrix, N)
print(PMIMatrix)
```

```
[[ 0.06938949  0.06130258 -1.13210017]
 [-0.32087251 -0.27871859  1.67836234]]
```

7) all values less than 0 are resulted into zero in PPMI matrix

```
def ppmi_matrix(PMIMatrix):
    (row,col) = PMIMatrix.shape

    PPMIMatrix = np.zeros((row,col))

    for i in range(row):
        for j in range(col):
            pmi = PMIMatrix[i,j]
            if(pmi < 0):
                PPMIMatrix[i,j] = 0
            else:
                PPMIMatrix[i,j] = pmi
    return PPMIMatrix
print(ppmi_matrix(PMIMatrix))
```

```
[[0.06938949 0.06130258 0.        ]
 [0.         0.         1.67836234]]
```

10) add 2 smoothing has been performed by adding 2 in every term and PMI matrix is calculated.

```
def pmi_matrix_smoothed(TermContextMatrix_smoothed, N):

    (row,col) = TermContextMatrix_smoothed.shape

    PMIMatrix_smoothed = np.zeros((row,col))

    for i in range(row):
        for j in range(col):
            pij = TermContextMatrix_smoothed[i,j] / N
            pw = p_information[i] / N
            pc = p_context[j] / N
            try:
                PMIMatrix_smoothed[i,j] = math.log((pij/(pw*pc)),2)
            except:
                PMIMatrix_smoothed[i,j] = 0
    return PMIMatrix_smoothed

PMIMatrix_smoothed = pmi_matrix(TermContextMatrix_smoothed, N)
print(PMIMatrix_smoothed)
```

```
[[ 0.12485041  0.1031523  -0.91952598]
 [-0.16857633 -0.19696361  1.81727949]]
```

```
def ppmi_matrix_smoothed(PMIMatrix_smoothed):
    (row,col) = PMIMatrix_smoothed.shape

    PPMIMatrix_smoothed = np.zeros((row,col))

    for i in range(row):
        for j in range(col):
            pmi = PMIMatrix_smoothed[i,j]
            if(pmi < 0):
                PPMIMatrix_smoothed[i,j] = 0
            else:
                PPMIMatrix_smoothed[i,j] = pmi
    return PPMIMatrix_smoothed
print(ppmi_matrix_smoothed(PMIMatrix_smoothed))
```

```
[[0.12485041 0.1031523  0.        ]
 [0.         0.         1.81727949]]
```

Problem 4:

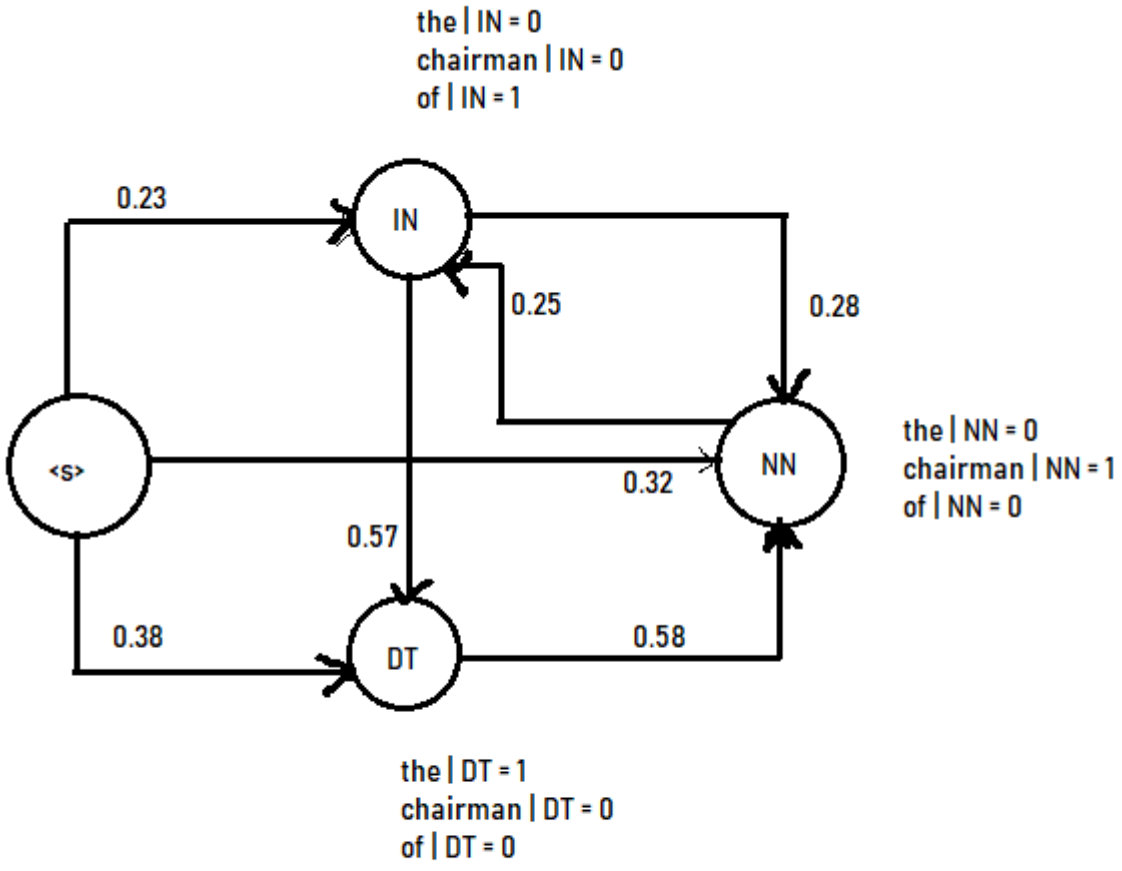
1. Create the Hidden Markov Model (HMM) and show (a) the transition probabilities and (b) observation likelihoods in each state that will be reached by sentences S1 and S2 after 3 time-steps. Present only the transition and observation likelihoods in the states reached after three steps. ***TOTAL: 5 points***

2. Create the Viterbi table for each sentence and populate it entirely. ***TOTAL: 10 points***

3. What is the probability of assigning the tag sequence for each of the sentences.? ***TOTAL: 10 points***

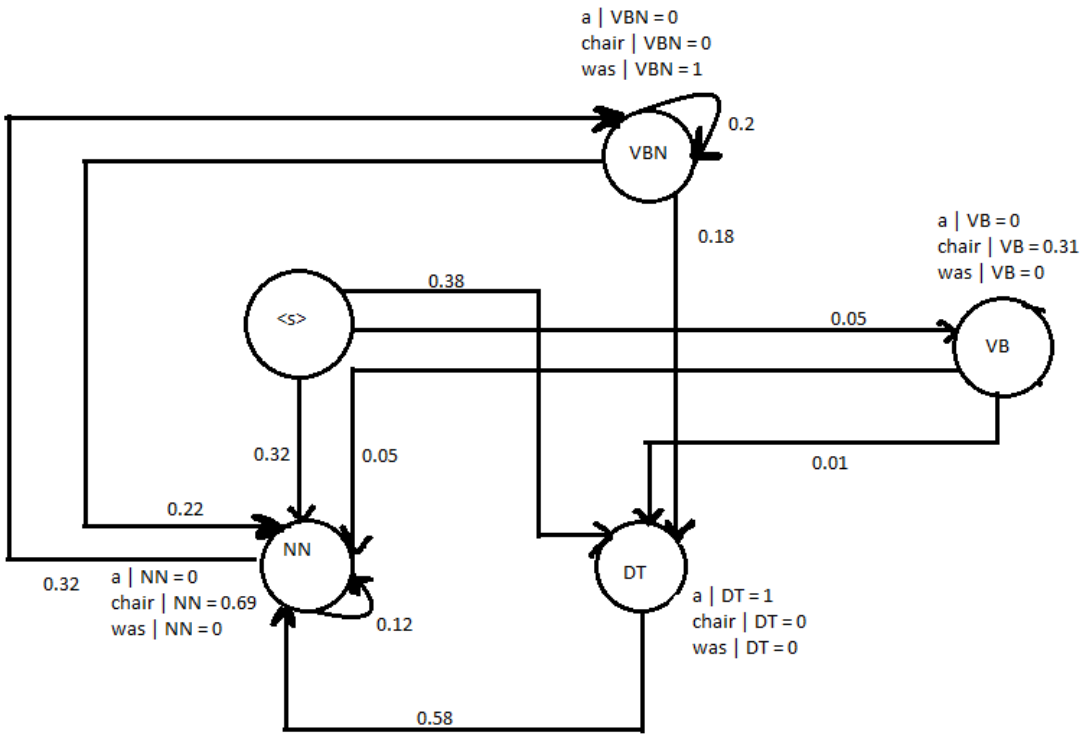
4. Execute Stanford POS-tagger (available from: https://nlp.stanford.edu/static/software/tagger.shtml) on both sentences. Which POS tags were assigned by the Stanford POS-tagger more accurately? Explain. ***TOTAL: 5 points***

1)



Hidden Markov Model for S1

The chairman of the board is completely bold.



Hidden Markov Model for s2

A chair was found in the middle of the road.

2)

Viterbi table for first sentence: The chairman of the board is completely bold.

The/DT chairman/NN of/IN the/DT board/NN is/VBZ completely/RB bold/JJ

</s>	0	0	0	0	0	0	0	0	0
IN	0.23	0	0	1*0.2204*0.25= 0.0551	0	0	0	0	0
RB	0.1	0	0	0	0	0	0	8*10^-4 * 0.15 = 1.2*10^-4	0
JJ	0.11	0	0	0	0	0	0	0	1.2 * 10^-4 * 0.39 2.9*10^-5
VBN	0	0	0	0	0	0	0	0	0
VBZ	0	0	0	0	0	0	0.016 * 0.05 = 8*10^-4	0	0
VB	0.05	0	0	0	0	0.12 * 0.0314 * 0 = 0	0	0	0
NN	0.32	0	1*0.38*0.58 = 0.2204	0	0	0.88 * 0.0314 * 0.12 = 0.016	0	0	0
DT	0.38	0.38 * 1 = 0.38	0	0	1*0.0551*0.57= 0.0314		0	0	0
<s>	0	0	0	0	0	0	0	0	0
	<s>	The	Chairman	of	the	board	is	completely	bold

Viterbi table for: A chair was found in the middle of the road.

A/DT chair/NN was/VBN found/VBN in/IN the/DT middle/NN of/IN the/DT road/NN

IN	0.23	0	0	0	0	1.04*10^-3	0	0	5.645 * 10^-5	0	0
RB	0.1	0	0	0	0	0	0	0	0	0	0
JJ	0.11	0	0	0	0	0	0	0	0	0	0
VBN	0	0	0	0.32 * 1* 0.152 = 0.048	9.5*10^-3	0	0	0	0	0	0
VBZ	0	0	0	0	0	0	0	0	0	0	0
VB	0.05	0	0.31 * 0.38 * 0 = 0	0	0	0	0	0	0	0	0
NN	0.32	0	0.69* 0.38 * 0.57 = 0.152	0	1*10^-4	0	0	2.258 * 10^-4	0	0	1.8*10^-5
DT	0.38	1*0.38 = 0.38		0			5.9 * 10^-4	0	0	3.217* 10^-5	0
<s>	0	0	0	0	0	0	0	0	0	0	0
	<s>	A	chair	was	found	in	the	middle	of	the	road

3)

The	Chairman	Of	The	Board	Is	Completely	bold
DT	NN	IN	DT	NN	VBZ	RB	JJ

The probability of the sentence: 2.9 * 10^-5

A	Chair	Was	Found	In	The	Middle	Of	The	road
DT	NN	VBN	VBN	IN	DT	NN	IN	DT	NN

The probability of the sentence: 1.8 * 10^-5

4)

Explanation:

Stanford POS appears to be more accurate. Because of the sentence S2 ‘was’ is tagged as VBD. However with Viterbi POS, tags it as VBN. (I used online parse nlp.stanford.edu:8080/parser/index.jsp

Not secure | nlp.stanford.edu:8080/parser/index.jsp

Stanford Parser

Please enter a sentence to be parsed:

The chairman of the board is completely bold

Language: English ▾ Sample Sentence Parse

Your query

The chairman of the board is completely bold

Tagging

```
The/DT  chairman/NN  of/IN  the/DT  board/NN  is/VBZ  completely/RB  bold/JJ
```

Parse

```
(ROOT
  (S
    (NP
      (NP (DT The) (NN chairman))
      (PP (IN of)
        (NP (DT the) (NN board))))
    (VP (VBZ is)
      (ADJP (RB completely) (JJ bold)))))
```

Not secure | nlp.stanford.edu:8080/parser/index.jsp

Stanford Parser

Please enter a sentence to be parsed:

A chair was found in the middle of the road

Language: English ▾ Sample Sentence Parse

Your query

A chair was found in the middle of the road

Tagging

```
A/DT  chair/NN  was/VBD  found/VBN  in/IN  the/DT  middle/NN  of/IN  the/DT  road/NN
```