Assignment #4

4.1 List Unigram and Unigram from sentences.

Code

```
import re
from nltk.util import ngrams
from nltk.tokenize import word_tokenize
text = "Cherry blossom represents the nature of life and a season of renewal i
n Japanese culture. Last year, the season attracted nearly five million people
 and boosted the economy by about $2.7 billion, according to figures from Bloo
mberg"
text = text.replace('.',' ')

def find_ngrams(data,n):
    words = word_tokenize(data)
    regex_word = [w for w in words if re.search(r'[a-zA-Z0-9]+', w)]
    n_grams = list(ngrams(regex_word,n))
    return n_grams

print("Unigram = ",find_ngrams(text,1))
print("Bigram = ",find_ngrams(text,2))
```

Result

```
D:\NLP>c:/python38/python.exe d:/NLP/4-1.py
Unigram =  [('Cherry',), ('blossom',), ('represents',), ('the',), ('nature',), ('of',), ('life',), (
'and',), ('a',), ('season',), ('of',), ('renewal',), ('in',), ('Japanese',), ('culture',), ('Last',)
, ('year',), ('the',), ('season',), ('attracted',), ('nearly',), ('five',), ('million',), ('people',
), ('and',), ('boosted',), ('the',), ('economy',), ('by',), ('about',), ('2',), ('7',), ('billion',)
, ('according',), ('to',), ('figures',), ('from',), ('Bloomberg',)]
Bigram =  [('Cherry', 'blossom'), ('blossom', 'represents'), ('represents', 'the'), ('the', 'nature'
), ('nature', 'of'), ('of', 'life'), ('life', 'and'), ('and', 'a'), ('a', 'season'), ('season', 'of'
), ('of', 'renewal'), ('renewal', 'in'), ('in', 'Japanese'), ('Japanese', 'culture'), ('culture', 'L
ast'), ('Last', 'year'), ('year', 'the'), ('the', 'season'), ('season', 'attracted'), ('attracted',
'nearly'), ('nearly', 'five'), ('five', 'million'), ('million', 'people'), ('people', 'and'), ('and'
, 'boosted'), ('boosted', 'the'), ('the', 'economy'), ('economy', 'by'), ('by', 'about'), ('about',
'2'), ('2', '7'), ('7', 'billion'), ('billion', 'according'), ('according', 'to'), ('to', 'figures')
, ('figures', 'from'), ('from', 'Bloomberg')]
```

4.2 Predict the possible word by measuring n-gram probability.

Code

```
import re
from nltk.util import ngrams
from nltk.tokenize import word_tokenize
from nltk.collocations import BigramCollocationFinder,BigramAssocMeasures
import operator

n=2
test_sentence = "I love X"
sentences = ["I love you!","I love mom.","I love you so much","I love dog.
","I love mom so much.","John love you so bad."]

def find_ngrams(data,n):
    n_grams = []
    for i in data:
        words = word_tokenize(i)
        regex_word = [w for w in words if re.search(r'[a-zA-Z0-9]+', w)]
        for j in ngrams(regex_word,n):
            n_grams.append(j)
    return n_grams

def Unigram_Bigram_count(Unigram,Bigram):
    bigramCounts = {}
    unigramCounts = {}

    for i in Unigram:
        if i in unigramCounts:
            unigramCounts[i] += 1
        else:
            unigramCounts[i] = 1

    for j in Bigram:
        if (j[0],j[1]) in bigramCounts:
            bigramCounts[(j[0],j[1])] +=1
        else:
            bigramCounts[(j[0],j[1])] =1

    return unigramCounts, bigramCounts


def calcBigramProb(listOfBigrams, unigramCounts, bigramCounts,word):
    listOfProb = {}
    max_Prob_value = 0
    max_Prob_word = ""
    for bigram in listOfBigrams:
```

```
        if(bigram[0] == word):
            listOfProb[bigram] = (bigramCounts.get(bigram))/(unigramCounts
.get(word))
            if(listOfProb[bigram] > max_Prob_value):
                max_Prob_value = listOfProb[bigram]
                max_Prob_word = bigram[1]
    return listOfProb , max_Prob_word

test_sentence2 = test_sentence.split()
listOfUnigrams = [i[0] for i in find_ngrams(sentences,1)]
listOfBigrams = find_ngrams(sentences,n)

unigramCounts, bigramCounts = Unigram_Bigram_count(listOfUnigrams,listOfBi
grams)
print("\n All the possible Bigrams are ",)
print(listOfBigrams)

print("\n Bigrams frequency ")
print(bigramCounts)

print("\n Unigrams frequency ")
print(unigramCounts)

bigramProb , max_Probability_word = calcBigramProb(listOfBigrams, unigramC
ounts, bigramCounts,test_sentence2[1])

print("\n Bigrams probability ")
print(bigramProb)

print(f"\n{test_sentence} : {test_sentence2[2]} should be '{max_Probabilit
y_word}'")
```

Result

```
D:\NLP>c:/python38/python.exe d:/NLP/4-2.py

 All the possible Bigrams are
[('I', 'love'), ('love', 'you'), ('I', 'love'), ('love', 'mom'), ('I', 'love'), ('love', 'you'), ('y
ou', 'so'), ('so', 'much'), ('I', 'love'), ('love', 'dog'), ('I', 'love'), ('love', 'mom'), ('mom',
'so'), ('so', 'much'), ('John', 'love'), ('love', 'you'), ('you', 'so'), ('so', 'bad')]

 Bigrams frequency
{('I', 'love'): 5, ('love', 'you'): 3, ('love', 'mom'): 2, ('you', 'so'): 2, ('so', 'much'): 2, ('lo
ve', 'dog'): 1, ('mom', 'so'): 1, ('John', 'love'): 1, ('so', 'bad'): 1}

 Unigrams frequency
{'I': 5, 'love': 6, 'you': 3, 'mom': 2, 'so': 3, 'much': 2, 'dog': 1, 'John': 1, 'bad': 1}

 Bigrams probability
{('love', 'you'): 0.5, ('love', 'mom'): 0.3333333333333333, ('love', 'dog'): 0.16666666666666666}

I love X : X should be 'you'
```