Experiment 2

Aim

Generate bigrams and trigrams from a given corpus and calculate probability of a sentence.

Theory

Probability of a sentence can be calculated by the probability of sequence of words occurring in it. We can use Markov assumption that the probability of a word in a sentence depends on the probability of the word occurring just before it. Such a model is called first order Markov model or the bigram model.

$$P(W_n | W_{n-1}) = P(W_{n-1}, W_n)/P(W_{n-1})$$

Here, W_n refers to the word token corresponding to the nth word in a sequence.

A combination of words forms a sentence. However, such a formation is meaningful only when the words are arranged in some order.
Eg: Sit I car in the
Such a sentence is not grammatically acceptable. However some perfectly grammatical sentences can be nonsensical too!
Eg: Colorless green ideas sleep furiously
One easy way to handle such unacceptable sentences is by assigning probabilities to the strings of words i.e, how likely the sentence is.

Probability of a sentence
If we consider each word occurring in its correct location as an independent event, the probability of the sentences is : P(w(1), w(2)..., w(n-1), w(n))

Using chain rule:
=**P(**w(1)**)** * **P(**w(2) | w(1)**)** * **P(**w(3) | w(1)w(2)**)** ... **P(**w(n) | w(1)w(2)…w(n-1)**)**

Bigrams
We can avoid this very long calculation by approximating that the probability of a given word depends only on the probability of its previous words. This assumption is called Markov assumption and such a model is called Markov model- bigrams. Bigrams can be generalized to the n-gram which looks at (n-1) words in the past. A bigram is a first-order Markov model.
Therefore ,
P(w(1), w(2)..., w(n-1), w(n)**)**= **P(**w(2)|w(1)**)** **P(**w(3)|w(2)**)** …. **P(**w(n)|w(n-1)**)**

We use (eos) tag to mark the beginning and end of a sentence.
A bigram table for a given corpus can be generated and used as a lookup table for calculating probability of sentences.

Eg: Corpus – (eos) You book a flight (eos) I read a book (eos) You read (eos)

Bigram Table:

	(eos)	you	book	a	flight	I	read
(eos)	0	0.5	0	0	0	0.25	0
you	0	0	0.5	0	0	0	0.5
book	0.5	0	0	0.5	0	0	0
a	0	0	0.5	0	0.5	0	0
flight	1	0	0	0	0	0	0
I	0	0	0	0	0	0	1
read	0.5	0	0	0.5	0	0	0

P((eos) you read a book (eos)**)**
= **P(**you|eos**)** * **P(**read|you**)** * **P(**a|read**)** * **P(**book|a**)** * **P(**eos|book**)**
= 0.5 * 0.5 * 0.5 * 0.5 * 0.5
= 0.03125

Code

```
import nltk
nltk.download('punkt')
text = "(eos) You book a flight (eos) I read a book (eos) You read (eos)"

unigram = text.split()
bigrm = list(nltk.bigrams(text.split()))
trigrm = list(nltk.trigrams(text.split()))

bigram = []
for element in bigrm:
  lst = ' '.join(map(str, element))
  bigram.append(lst)
bigram

fdist = nltk.FreqDist(bigram)
freq2 = []
for k,v in fdist.items():
    freq2.append((k,v))
freq2

fdist = nltk.FreqDist(unigram)
freq = []
for k,v in fdist.items():
    freq.append((k,v))

freq

txt = "(eos) You book a flight (eos)"
output = list(nltk.bigrams(txt.split()))

oput = []
for element in output:
  lst = ' '.join(map(str, element))
  oput.append(lst)

oput
```

```
prob = 1
for item in oput:
  print(item)
  word = item.split()[0]
  for element in freq2:
    if(item == element[0]):
      num = element[1]
      break
  for element in freq:
    if(word == element[0]):
      den = element[1]
      break
  print(item)
  print(num/den)
  prob = prob * num / den
prob
```

Output

```
Corpous : (eos) You book a flight (eos) I read a book (eos) You read (eos)
Sen : (eos) You read a book (eos)
(eos) You
0.5
You read
0.5
read a
0.5
a book
0.5
book (eos)
0.5
Probability of the Sentence = 0.03125
```

Conclusion

Thus, the probability for given bigrams and trigrams are computed.