**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.



Here, Wn 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)

**Code & Output:**

| corpus = "You book a flight. I read a book. You read."  import nltk  nltk.download('punkt') |
| --- |

| def preprocess(d):  d=d.lower()  d="eos "+ d  d=d.replace("."," eos")  return d  d=preprocess(corpus)  print("Preprocessed Data corpus = \n",d) |
| --- |

| Preprocessed Data corpus =  eos you book a flight eos i read a book eos you read eos |
| --- |

| from nltk import word\_tokenize  def generate\_tokens(d):  tokens = word\_tokenize(d)  return tokens  tokens=generate\_tokens(d)  distinct\_tokens = list(set(sorted(tokens)))  print("Tokens in the corpus = \n",distinct\_tokens) |
| --- |

| Tokens in the corpus =  ['book', 'a', 'flight', 'you', 'eos', 'read', 'i'] |
| --- |

| def generate\_tokens\_freq(tokens):  dct={}  for i in tokens:  dct[i]=0  for i in tokens:  dct[i]+=1  return dct  dct=generate\_tokens\_freq(tokens)  print("Frequency of each tokens = ")  for i in dct.items():  print(i[0],"\t:" , i[1]) |
| --- |

| Frequency of each tokens =  eos : 4  you : 2  book : 2  a : 2  flight : 1  i : 1  read : 2 |
| --- |

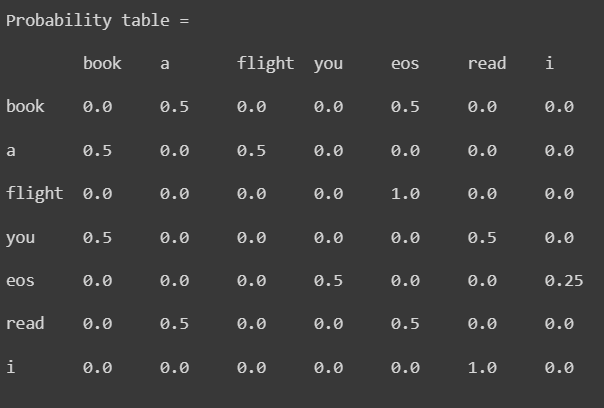
| def generate\_ngrams(tokens,k):  l=[]  i=0  while(i<len(tokens)):  l.append(tokens[i:i+k])  i=i+1  l=l[:-1]  return l  bigram = generate\_ngrams(tokens,2)  print("N-grams generated (Here n is 2) = ")  for i in bigram:  print(i) |
| --- |

| N-grams generated (Here n is 2) =  ['eos', 'you']  ['you', 'book']  ['book', 'a']  ['a', 'flight']  ['flight', 'eos']  ['eos', 'i']  ['i', 'read']  ['read', 'a']  ['a', 'book']  ['book', 'eos']  ['eos', 'you']  ['you', 'read']  ['read', 'eos'] |
| --- |

| def generate\_ngram\_freq(bigram):  dct1={}  for i in bigram:  st=" ".join(i)  dct1[st]=0  for i in bigram:  st=" ".join(i)  dct1[st]+=1  return dct1  dct1=generate\_ngram\_freq(bigram)  print("Frequency of n-grams = ")  for i in dct1.items():  print(i[0], ":", i[1]) |
| --- |

| Frequency of n-grams =  eos you : 2  you book : 1  book a : 1  a flight : 1  flight eos : 1  eos i : 1  i read : 1  read a : 1  a book : 1  book eos : 1  you read : 1  read eos : 1 |
| --- |

| def find1(s,dct1):  try:  return dct1[s]  except:  return 0  def print\_probability\_table(distinct\_tokens,dct,dct1):  n=len(distinct\_tokens)  l=[[]\*n for i in range(n)]  for i in range(n):  denominator = dct[distinct\_tokens[i]]  for j in range(n):  numerator = find1(distinct\_tokens[i]+" "+distinct\_tokens[j],dct1)  l[i].append(float("{:.3f}".format(numerator/denominator)))  return l  print("Probability table = \n")  probability\_table=print\_probability\_table(distinct\_tokens,dct,dct1)  n=len(distinct\_tokens)  print("\t", end="")  for i in range(n):  print(distinct\_tokens[i],end="\t")  print("\n")  for i in range(n):  print(distinct\_tokens[i],end="\t")  for j in range(n):  print(probability\_table[i][j],end="\t")  print("\n") |
| --- |



| input\_text = "You read a book."  p = preprocess(input\_text)  print("Preprocessed Text = \n",p,"\n")  t = generate\_tokens(p)  print("Tokens Generated = \n",t,"\n")  n = generate\_ngrams(t,2)  print("N-grams Generated = \n= ",n) |
| --- |

| Preprocessed Text =  eos you read a book eos  Tokens Generated =  ['eos', 'you', 'read', 'a', 'book', 'eos']  N-grams Generated =  = [['eos', 'you'], ['you', 'read'], ['read', 'a'], ['a', 'book'], ['book', 'eos']] |
| --- |

| for i in n:  print("'{}'".format(' '.join(i)), end=", ")  print("\n\n"+'\033[1m'+"Calculate bigram probability"+'\033[0m')  s=1  dct2={}  for i in n:  dct2[" ".join(i)]=0    for i in n:  k=distinct\_tokens.index(i[0])  m=distinct\_tokens.index(i[1])  dct2[" ".join(i)]=probability\_table[k][m]  print("P('{}')\t= ".format(' '.join(i)),probability\_table[k][m])  s\*=probability\_table[k][m]  print("\n"+'\033[1m'+ "Calculate Probability of the sentence"+'\033[0m')  print(f"P('{input\_text}') \n= ",end="")  x=dct2.popitem()  for i in dct2:  print(f"P('{i}')", end=" \* ")  print(f"P('{x[0]}')\n= ", end='')  for i in dct2:  print(dct2[i], end=" \* ")  print(x[1],"\n=",s)  print("\n"+'\033[1m'+f"Probability('{input\_text}') = "+"{:.5f}".format(s)) |
| --- |

| 'eos you', 'you read', 'read a', 'a book', 'book eos',  Calculate bigram probability  P('eos you') = 0.5  P('you read') = 0.5  P('read a') = 0.5  P('a book') = 0.5  P('book eos') = 0.5  Calculate Probability of the sentence  P('You read a book.')  = P('eos you') \* P('you read') \* P('read a') \* P('a book') \* P('book eos')  = 0.5 \* 0.5 \* 0.5 \* 0.5 \* 0.5  = 0.03125  Probability('You read a book.') = 0.03125 |
| --- |

| trigram = generate\_ngrams(tokens,3)  print("N-grams generated (Here n is 3) = ")  for i in trigram:  print(i) |
| --- |

| N-grams generated (Here n is 3) =  ['eos', 'you', 'book']  ['you', 'book', 'a']  ['book', 'a', 'flight']  ['a', 'flight', 'eos']  ['flight', 'eos', 'i']  ['eos', 'i', 'read']  ['i', 'read', 'a']  ['read', 'a', 'book']  ['a', 'book', 'eos']  ['book', 'eos', 'you']  ['eos', 'you', 'read']  ['you', 'read', 'eos']  ['read', 'eos'] |
| --- |

| dct2=generate\_ngram\_freq(trigram)  print("Frequency of n-grams = ")  for i in dct2.items():  print(i[0], ":", i[1]) |
| --- |

| Frequency of n-grams =  eos you book : 1  you book a : 1  book a flight : 1  a flight eos : 1  flight eos i : 1  eos i read : 1  i read a : 1  read a book : 1  a book eos : 1  book eos you : 1  eos you read : 1  you read eos : 1  read eos : 1 |
| --- |

**Conclusion:** In this experiment, we have implemented Bigrams and Trigrams using python.