Lab 4

**English Language Model using N-gram probabilistic model**

1. Write a python method which create a **uni-gram** model using a dictionary of English vocabulary collected from any corpus of Project Gutenberg.
2. Write a python method which create a **bi-gram** model using same dictionary used in the above method.
3. Write a python method which create a **Tri-gram** model using same dictionary used in the above method.
4. Write a method to generate a sentence of 10 words from some initial words using previously learned n-gram models.
5. Write a simple test drive python code to test the above methods

import nltk  
from nltk.util import ngrams  
from nltk.corpus import stopwords  
from nltk.tokenize import word\_tokenize  
from collections import defaultdict  
import random  
corpus = nltk.corpus.gutenberg.words('austen-emma.txt')  
  
def unigram():  
 text = nltk.corpus.gutenberg.raw("shakespeare-caesar.txt")  
  
 # Create a unigram model  
 unigram\_model = nltk.FreqDist(text.split())  
  
 # Generate a sentence  
 sentence = ""  
 for i in range(0, 9):  
 for word in random.choices(unigram\_model.most\_common(10)):  
 sentence += word[0] + " "  
 return sentence  
  
  
def create\_ngram\_model(n, corpus):  
   
 *"""  
 Create an n-gram model using a dictionary of English vocabulary  
 collected from any corpus of Project Gutenberg.  
  
 :param n: The value of n for the n-gram model.  
 :param corpus: A list of words representing the corpus.  
 :return: A dictionary representing the n-gram model.  
 """* ngrams = list(nltk.ngrams(corpus, n))  
 model = defaultdict(lambda: defaultdict(lambda: 0))  
 for ngram in ngrams:  
 model[ngram[:-1]][ngram[-1]] += 1  
 for w1\_w2 in model:  
 total\_count = float(sum(model[w1\_w2].values()))  
 for w3 in model[w1\_w2]:  
 model[w1\_w2][w3] /= total\_count  
 return model  
  
  
def generate\_sentence(model, initial\_words, num\_words):  
 *"""  
 Generate a sentence of a given number of words from some initial words  
 using a previously learned n-gram model.  
  
 :param initial\_words: A tuple of initial words to start the sentence with.  
 :param num\_words: The number of words to generate in the sentence.  
 :return: A list of words representing the generated sentence.  
 """* sentence = list(initial\_words)  
 for i in range(num\_words):  
 next\_word = random.choices(list(model[tuple(sentence[-len(initial\_words):])].keys()),  
 list(model[tuple(sentence[-len(initial\_words):])].values()))[0]  
 sentence.append(next\_word)  
 return sentence  
  
# Load the text corpus  
  
# Test drive code  
unigram\_model = unigram()  
bigram\_model = create\_ngram\_model(2, corpus)  
trigram\_model = create\_ngram\_model(3, corpus)  
  
initial\_words = random.choices(corpus)  
initial\_words2 = ("the", "ladies")  
  
s2 = generate\_sentence(bigram\_model, initial\_words, 10)  
s3 = generate\_sentence(trigram\_model, initial\_words2, 10)  
print(''.join(unigram\_model))  
print(' '.join(s2))  
print(' '.join(s3))

output:

I And I the And to and you to

good specimens of his children might talk of a mixture of

the ladies in the rank of life ." After this , that