Lab 02 NLTK: Language Models

Task 1: Bigrams & Trigrams

Start with the frequency distribution of text from the corpuses used from lab01.

Acquire the unique sorted words (without punctuations) from text1 and store them in a list called **words**. Example of text1 is given below (but you can use stop words stop punctuations also)

```
words = sorted(set(text1))[280:]
```

You can browse through this list with any set of variations. For example:

```
longwords = [w for w in words if len(w) > 16]

high_freq = [w for w in words if fdist1[w] > 500]

high_freq_idf = [w for w in words if IDF(text1, w) > 1]

eign_words = [w for w in words if w.endswith('eign')
```

You can also use functions such as w.startswith(a), w.endswith(z), substring in w, w.islower(), w.isupper(), w.istitle() for camel case, w.isalpha(), w.isdigit(), etc. with the conditions. Note you can also use regular control structures for these activities:

```
for w in words:
    if w.endswith('eign'):
        print(w)
```

Moving towards language models, we would first convert a given text corpus to its bigram model using:

```
list(bigrams(text1))
```

You can view the first 10 by appending [:10] to the given command. To view the most frequent bigrams in the text, you can use the collocations feature of NLTK as:

```
text1.collocations()
```

For any model higher than a bigram, you can use ngrams from NLTK as:

```
from nltk.util import ngrams

list(ngrams(text1, 3))
```

For usage of trigram with collocations, you can use:

```
from nltk.collocations import *

TrigramCollocationFinder.from_words(text1).nbest(TrigramAssocMeasures().pmi, 10)
```

Now, fill the following table for the most common bigrams and trigrams in the given texts:

	Text1	Text2	Text3
10 frequently occuring Bigrams			
5 frequently occuring Trigrams			

Number of words with length > 16			
Number of words with frequency > 500			
Number of words ending in “ed”			

Task 2: Accessing Corpora

NLTK contains many electronic books stored from Project Gutenberg. You can view these books by running

```
nltk.corpus.gutenberg.fileids()
```

As an example, to view the words in Shakespeare Ceaser, you will provide:

```
gutenberg_sc = nltk.corpus.gutenberg.words('shakespeare-caesar.txt')
```

To view the Brown corpus words, you will likewise use:

```
from nltk.corpus import brown

brown.words()
```

Most of the time, you will be needing to provide custom data manually or through web scraping. For this, you can load your own data to NLTK for evaluation purpose. Since you will be having multiple documents in the corpus, it is better that some folder structuring is used. You can make it consistent with the default nltk_data folder which you have created so far. This nltk_data folder is created when you ran nltk.download() in last lab. Alternatively, you can specify any other location of your choosing. To load the data, you can then use:

```
psh_raw = nltk.data.load('~/nltk_data/corpora/omar/peshawar.txt', format ='raw')

psh_txt = nltk.data.load('~/nltk_data/corpora/omar/peshawar.txt', format ='text')
```

Raw data is machine readable and in byte form, whereas text data is human readable. To test, you can proceed with reading your file for processing in many applications. The following will help you get the bigrams and trigrams from your file.

```
from nltk.util import ngrams

words = nltk.word_tokenize(psh_txt)

psh_bigrams = list(ngrams(words, 2))

psh_trigrams = list(ngrams(words, 3))
```

The rest of the processing can be done to your requirements. However, to read multiple files, you would need to use NLTK’s Plain Text Corpus Reader as:

```
from nltk.corpus import PlaintextCorpusReader

corpus_root = os.path.expanduser('/home/omar/nltk_data/corpora/omar/')

corpus = PlaintextCorpusReader(corpus_root, '.*', encoding='latin1')

wordlist = corpus.words()

bigramlist = list(ngrams(wordlist,2))
```

Task 3: Generating Random Text with Bigrams

For this exercise, you can take any text that you want (from an existing corpus, or your own). You can think of some poetry or chatbot text etc. We would be needing a generative model to generate text. Access your corpus through PlaintextCorpusReader() and generate a list of bigrams. Then, prepare a conditional frequency distribution from this corpus as follows:

```
cfd = nltk.ConditionalFreqDist(bigramlist)
```

We would now need a statistical generator which will select a word on the basis of highest probability:

```
def generate_model(cfdist, word, num):

    for i in range(num):

        print(word, end=' ')

        word = cfdist[word].max()
```

Finally, we will call this model to generate 10 words from the list

```
generate_model(cfd, 'Today', 10)
```

Later on, we will look at some more advanced tools for generating high quality text. Experiment with the above to generate some text and fill the following information:

Number of Words in Corpus	30 word generated sentence