Intermediate Report:

Vivian Tan (vyt73)

The overall project is based on applications of language models, more specifically text generation based on n-grams. Random text generation can be useful for designers to use in different settings such as web applications, site templates, and topography demos.

I will be using the n-gram language model to complete this project, starting with bigrams and then moving to trigrams if time permits. I will also use Maximum Likelihood Estimation (MLE) to calculate the probabilities used to select subsequent words. To store the texts, I will be using lists of strings/tokens for each work. Eventually, I hope to combind different works from the same author into a single list.

I'll use texts from Project Gutenberg as my corpus, more specifically, from these Authors: (may remove/add authors based on progress)

- Jane Austen
- G.K. Chesterton
- William Shakespeare
- Arthur Conan Doyle
- Fyodor Dostoyevsky

Below I will show and discuss my initial results.

Initial results:

This project is based on applications of language models, more specifically text generation based on n-grams. The end goal is to generate a random text based on two different author's writing styles, but for this initial report we will:

- explore different methods to generate a random text from a single work
- explore different techniques to get works from URLs

Generating text:

First, let's generate a random text from a single work:

In [1]:

```
# Imports:
import nltk
import random
from nltk.corpus import brown
from nltk import word_tokenize
from nltk.util import ngrams
import random
```

Using bigrams and MLE by hand: (based on hw code)

This first technique is from one of the homework assignments. It generates text by counting the bigrams for a text and calculating the probabilities for each then using these to generate the next word. It could easily be modified to account for trigrams.

In [2]:

```
def count_and_prob(corpus, count, prob):
        count2 = {}
        # Count bigrams and instantiate probability for each word to 0
        for word1, word2 in nltk.bigrams(corpus):
            if word1 not in count:
                count[word1] = { }
                count2[word1] = 0
                prob[word1] = {}

            if word2 not in count[word1]:
                #print(word1, word2)
                count[word1][word2] = 1
                count2[word1] += 1
            else:
                count[word1][word2] += 1
                count2[word1] += 1

        # Recall: To estimate probabilities of bigrams:
        # P(word2|word1) = count(word1, word2)/count(word1, ...)

        # for each word, calculate probability of bigram(s)
        for word1 in count:
                for word2 in count[word1]:
                        # print(word1, word2,':', count[word1][word2]/count2[wor
d1])
                        prob[word1][word2] = count[word1][word2]/count2[word1]

        return count, prob
```

In [3]:

```
# get the next word based on probabilities
def gen_next_word(word, prob_dict):
        boundary = []
        words = []
        count = 0

        # Generate random number
        dart = random.random()

        if(len(prob_dict) == 0): return 'ERROR'

        # Get probabilities and words
        for w in prob_dict[word]:
                # get first probability and first word
                boundary.append(prob_dict[word][w])
                words.append(w)
                count += 1

        #print('boundary:',boundary)
        #print('words:', words)

        # return the next word based on the dart value and boundaries
        # if there is only one option, return the word:
        if count == 1:
                return words[0]
        else:
                #if there are more options, find the minimum
                #sort lists together
                boundary, words = (list(t) for t in zip(*sorted(zip(boundary, wo
rds))))

                # go through boundaries, starting with smallest
                # (Note: though this assignment could be completed w/a simple if
else,
                # I wanted to try and write code that could work with trigrams a
nd up)
                total = 0;
                for i in range(0,len(boundary)):
                        if i==0:
                                total = boundary[i]
                                if dart<=total:
                                        return words[i]
                        # add previous probabilities
                        elif dart<=(boundary[i]+total):
                                        return words[i]
                        else:
                                total += boundary[i]
                                # print('look again')
```

Let's test out the text generation with "Emma":

In [4]:

```
emma = nltk.corpus.gutenberg.words('austen-emma.txt')
emma[:10]
```

Out[4]:

```
['[', 'Emma', 'by', 'Jane', 'Austen', '1816', ']', 'VOLUME', 'I', 'C
HAPTER']
```

In [5]:

```
start = 'I'

count_dict = {}
prob_dict = {}
count_dict, prob_dict = count_and_prob(emma, count_dict, prob_dict)
next_word = start
print('\n\nGenerating random text: Emma\n')
for i in range(0,50):
        print(next_word, end =' ')
        next_word = gen_next_word(next_word, prob_dict)
```

```
Generating random text: Emma

I will laugh . Her silence . Wingfield considers the gentleman ' s a
ccommodation and Mr . Either bathing places in , her , I have been a
repetition of England -- to encroach .-- A situation -- and felt --
unaccountable as what letter from her through such feelings
```

These results show that we can generate a fairly coherent paragraph based on a single text using this technique. Let's try another technique:

Generating text using nltk:

Based on: http://www.katrinerk.com/courses/python-worksheets/language-models-in-python (http://www.katrinerk.com/courses/python-worksheets/language-models-in-python)

This technique requires less lines of code since it just uses some built-in nltk functions. However, this method is limited to only bigrams.

In [6]:

```
cfreq_emma_2gram = nltk.ConditionalFreqDist(nltk.bigrams(emma))
cprob_emma_2gram = nltk.ConditionalProbDist(cfreq_emma_2gram, nltk.MLEProbDist)
word = 'I'
for i in range(0,50):
    print(word, end=' ')
    word = cprob_emma_2gram[word].generate()
```

```
I have a system of the amiable and the deepest humiliation to sensat
ions had passed . They all your answer for her remains of the subjec
t , in feature works ." Emma ) that the _first_ with us with so prop
erly struck and you should he had been proposing to
```

This technique can also generate a paragraph based on a single text, but it doesn't quite seem as cohesive as the previous generated paragraph. It's sufficient for our purposes though. I think I may use the 2nd technique if I want to test text generation quickly and use the 1st technique with trigrams for the final result.

Now that we know that we can generate text from a single work, the next step would be to generate a single text from multiple works. To do that, we need to pull multiple works. Next we will explore different techniques to do that.

Getting multiple works online:

Through the nltk corpus:

The nltk gutenberg corpus comes with a few works, let's see what they have:

In [7]:

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

Out[7]:

```
['austen-emma.txt',
 'austen-persuasion.txt',
 'austen-sense.txt',
 'bible-kjv.txt',
 'blake-poems.txt',
 'bryant-stories.txt',
 'burgess-busterbrown.txt',
 'carroll-alice.txt',
 'chesterton-ball.txt',
 'chesterton-brown.txt',
 'chesterton-thursday.txt',
 'edgeworth-parents.txt',
 'melville-moby_dick.txt',
 'milton-paradise.txt',
 'shakespeare-caesar.txt',
 'shakespeare-hamlet.txt',
 'shakespeare-macbeth.txt',
 'whitman-leaves.txt']
```

In [8]:

```
import re
t1 = nltk.corpus.gutenberg.words('austen-emma.txt')
t2 = nltk.corpus.gutenberg.words('austen-persuasion.txt')
t3 = nltk.corpus.gutenberg.words('austen-sense.txt')
```

Looks like this corpus doesn't have books from all the authors we want, or the complete works of the authors we do have. However, if we decide to scale our project down to these works, we could save each of the texts and then concatenate them later. Let's see if we can get books directly from the Gutenberg website:

Through URLS:

Austen books:

Emma

"https://www.gutenberg.org/files/158/158-0.txt (https://www.gutenberg.org/files/158/158-0.txt)"

Pride and Prejudice

"http://www.gutenberg.org/cache/epub/42671/pg42671.txt
(http://www.gutenberg.org/cache/epub/42671/pg42671.txt)"

Love and Friendship

"https://www.gutenberg.org/files/1212/1212-0.txt (https://www.gutenberg.org/files/1212/1212-0.txt)"

Mansfield Park

"https://www.gutenberg.org/files/141/141-0.txt (https://www.gutenberg.org/files/141/141-0.txt)"

Northanger Abbey

"https://www.gutenberg.org/files/121/121-0.txt (https://www.gutenberg.org/files/121/121-0.txt)"

Persuasion

"https://www.gutenberg.org/files/121/121-0.txt (https://www.gutenberg.org/files/121/121-0.txt)"

Sense and Sensibility

"http://www.gutenberg.org/cache/epub/161/pg161.txt
(http://www.gutenberg.org/cache/epub/161/pg161.txt)"

Lady Susan

"http://www.gutenberg.org/cache/epub/946/pg946.txt
(http://www.gutenberg.org/cache/epub/946/pg946.txt)"

In [9]:

```
from urllib import request

# Austen:
# Pride and Prejudice
url = "http://www.gutenberg.org/cache/epub/42671/pg42671.txt"
response = request.urlopen(url)
raw = response.read().decode('utf8')
raw[:75]
```

Out[9]:

```
'\ufeffThe Project Gutenberg eBook, Pride and Prejudice, by Jane Aus
ten, Edited\r\n'
```

We can get a text file using the url successfully! Let's see if we can quickly generate a text with it:

In [10]:

```
tokens = word_tokenize(raw)
```

In [11]:

```
cfreq_pride_2gram = nltk.ConditionalFreqDist(nltk.bigrams(tokens))
cprob_pride_2gram = nltk.ConditionalProbDist(cfreq_pride_2gram, nltk.MLEProbDist
)
word = 'I'
for i in range(0,50):
    print(word, end=' ')
    word = cprob_pride_2gram[word].generate()
```

```
I do all . It was no more than once . '' said Jane as her , the summ
er . '' she was forced to be attempted to meet , a servant , on my h
appiness , `` Not that best politeness on any useful acquaintance en
tirely deceived herself .
```

Looks like we can successfully generate a random text from a file pulled from a url, this will be useful in future work.

Conclusion

In this intermediate report we were able to:

- Generate random text from a single file using 2 techniques
- Retrieve works from the nltk Gutenberg corpus
- Retrieve a text file from a URL and generate a text from it

The next steps will be:

- Retrieve multiple texts from URLs
- Combine multiple books for each author
- Generate a text based on two texts/authors