Problem 1 (b)

Definitions

Prior to beginning our work, we load the requiste packages:

In [1]:
```
import math
import time

t0 = time.time()
```

We also load all 26 letters (*in lower case*) into an array for later use. This is done by importing a text file containing these letters

In [2]:
```
with open('../LowerCaseAlphabet.txt', 'r') as myFile:
    lowerAlpha = myFile.readlines()
```

and then stripping the return characters (\n) from each line

In [3]:
```
for i in range(len(lowerAlpha)):
    lowerAlpha[i] = lowerAlpha[i].replace('\n','')

lowerAlphaB = ''.join(str(x) for x in lowerAlpha)
```

Finally, we check to see if the alphabet imported properly,

In [4]:
```
print(lowerAlphaB)
```

```
abcdefghijklmnopqrstuvwxyz
```

as well as create an upper case version

In [5]:
```
upperAlpha = []
for x in lowerAlpha:
    upperAlpha.append(x.upper())

upperAlphaB = ''.join(str(x) for x in upperAlpha)
```

and check it

In [6]:
```
print(upperAlphaB)
```

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Load File

We begin our work by loading the text from the source file:

In [7]:
```
with open('../Text-Files/sawyer-ascii.txt', 'r') as myFile:
        tempData = myFile.readlines()
```

Then, we get some basic information about the data imported from the file

In [8]:
```
print(len(tempData))
```

```
8807
```

Now, convert the array of strings to a single string.

In [9]:
```
data = ''.join(str(x) for x in tempData)
```

Then find and store the length of the resulting string:

In [10]:
```
charCNT = len(data)
print(charCNT)
```

```
402665
```

Next, the length compare it to the combined length of all the strings in the initial array we got from importing the text file.

In [11]:
```
cnt = 0
for x in tempData:
    cnt = cnt + len(x)

print(cnt)
```

```
402665
```

Since the character counts are accurate, we can proceed.

Get Character List

First, we will obtain a list of all characters occuring in the text

In [12]:
```
myChars = list(set(data))

myChars2 = ''.join(str(x) for x in myChars)
print(myChars)
print(myChars2)
```

```
['m', 'r', 'q', '6', 'A', '"', 'i', '[', 'L', ':', 'K', 'D', 'b', 'R', 'X', 'y',
'0', 'e', 'T', '-', 'p', 'd', 'C', '$', 'H', "'", '2', '~', 'J', 'F', 'V', '/',
's', '+', 'U', '%', 'g', ' ', '*', '8', 'Y', '<', '9', 'Q', '5', ']', '7', 'P',
'v', 'l', 'S', '?', '.', 'z', 't', 'N', '\n', 'n', '>', ';', '&', 'o', '@', 'f',
'h', '(', ',', 'E', 'B', 'M', '!', 'W', 'a', 'O', 'I', 'w', ')', '4', '_', 'c',
'#', 'G', 'k', 'u', 'x', '3', 'j', '1']
mrq6A"i[L:KDbRXy0eT-pdC$H'2~JFV/s+U%g *8Y<9Q5]7PvlS?.ztN
n>;&o@fh(,EBM!WaOIw)4_c#Gkux3j1
```

Next, we remove our non-alphabetic characters from the list of characters to eliminate

In [13]:
```
for x in lowerAlphaB:
    myChars2 = myChars2.replace(x,'')

for x in upperAlphaB:
    myChars2 = myChars2.replace(x,'')

print(len(myChars2))
print(myChars2)
```

```
37
6"[:0-$'2~/+% *8<95]7?.
>;&@(,!)4_#31
```

and check the results

In [14]:
```
print(list(set(myChars2)))
print(len(list(set(myChars2))))
```

```
['!', '$', "'", '2', '~', '6', '"', '[', '\n', ':', '/', '>', '+', ')', '%', ' '
, '*', '8', '<', '9', ';', '4', '5', '_', '#', '&', '0', ']', '7', '@', '3', '('
, ',', '1', '?', '-', '.']
37
```

Clean the Text

Now, we can eliminate these characters from the text

In [15]:
```
#create copy of imported data
data2 = data

for x in myChars2:
    data2 = data2.replace(x,'')
```

and check to make sure the lengths have changed

In [16]:
```
print(len(data))
print(len(data2))
```

```
402665
307917
```

Last, we convert all letters in the text to lowercase

In [17]:
```
data2 = data2.lower()
```

Generate N-Grams

For the next part, we will need lists of Bi-Grams and Tri-Grams based off the lowercase english alphabet. We start with Bi-Grams

Bi-Grams

We begin with the array of lowercase alphabetic characters for the english language; however, we must first define an empty array to hold the Bi-Grams that we will generate. After which, we loop through the lowercase alphabet twice, joining each pair and appending the new pair to our array.

In [18]:
```
myBiGrams = []

for xx in lowerAlpha:
    for yy in lowerAlpha:
        temp = ''.join(str(xyz) for xyz in [xx,yy])
        myBiGrams.append(temp)

print(len(myBiGrams))
```

```
676
```

Checking that the number of Bi-Grams we generated is correct, we have

In [19]:
```
print(26*26)
```

```
676
```

Tri-Grams

Again, we begin with the array of lowercase alphabetic characters for the english language; however, we must first define an empty array to hold the Tri-Grams that we will generate. After which, we loop through the lowercase alphabet three times, joining each triplet and appending the new triplet to our array.

In [20]:
```
myTriGrams = []

for xx in lowerAlpha:
    for yy in lowerAlpha:
        for zz in lowerAlpha:
            temp = ''.join(str(xyz) for xyz in [xx,yy,zz])
            myTriGrams.append(temp)

print(len(myTriGrams))
```

```
17576
```

Checking that the number of Tri-Grams we generated is correct, we have

In [21]:
```
print(26*26*26)
```

```
17576
```

Quad-Grams

Again, we begin with the array of lowercase alphabetic characters for the english language; however, we must first define an empty array to hold the Quad-Grams that we will generate. After which, we loop through the lowercase alphabet four times, joining each quadruplet and appending the new quadruplet to our array.

In [22]:
```
myQuadGrams = []

for ww in lowerAlpha:
    for xx in lowerAlpha:
        for yy in lowerAlpha:
            for zz in lowerAlpha:
                temp = ''.join(str(wxyz) for wxyz in [ww,xx,yy,zz])
                myQuadGrams.append(temp)

print(len(myQuadGrams))
```

```
456976
```

Checking that the number of Tri-Grams we generated is correct, we have

In [23]:
```
print(26*26*26*26)
```

```
456976
```

Get Frequencies and Probabilities

Frequencies

To get the Frequencies of each letter, we first create an arrays to store them for Bi-Grams and Tri-Grams

In [24]:
```
counts = []
cntsBI= []
cntsTRI = []
cntsQUAD = []
```

and then go through the alphabet counting

In [25]:
```
for x in lowerAlpha:
    counts.append(data2.count(x))

print(len(counts))
print(counts)
```

```
26
[24352, 5221, 6873, 15302, 37080, 6270, 6841, 19997, 19642, 692, 3138, 12565, 74
44, 20959, 24325, 4950, 194, 16092, 18376, 29970, 9340, 2474, 8244, 387, 7032, 1
57]
```

and then go though the Bi-Grams counting

In [26]:
```
for x in myBiGrams:
    cntsBI.append(data2.count(x))

print(len(cntsBI))
```

```
676
```

and then the Tri-Grams counting

In [27]:
```
for x in myTriGrams:
    cntsTRI.append(data2.count(x))

print(len(cntsTRI))
```

```
17576
```

and then the Quad-Grams counting

In [28]:
```
for x in myQuadGrams:
    cntsQUAD.append(data2.count(x))

print(len(cntsQUAD))
```

```
456976
```

Character and N-Gram totals

Now, we can compute the probabilities. First, we store the number of characters in the cleaned text

In [29]:
```
charTOT = len(data2)

print(charTOT)
```

```
307917
```

which we check against the frequencies we just calculated

In [30]:
```
print(sum(counts))
```

```
307917
```

Then we compute the number of Bi-, Tri-, and Quad- Grams based on the total number of characters in the text

In [31]:
```
biTOT = math.floor(charTOT / 2)
triTOT = math.floor(charTOT / 3)
quadTOT = math.floor(charTOT / 4)
```

which gives

In [32]:
```
print(biTOT)
print(triTOT)
print(quadTOT)
```

```
153958
102639
76979
```

Probabilities

Again, we first create and empty array to hold the probabilities for single characters, as well as all our N-Grams

In [33]:
```
prbs = []
biPRBS = []
triPRBS = []
quadPRBS = []
```

then we loop through the list of frequencies, using them to create each probability

In [34]:
```
for x in counts:
    prbs.append(x / charTOT)

for x in cntsBI:
    biPRBS.append(x / biTOT)

for x in cntsTRI:
    triPRBS.append(x / triTOT)

for x in cntsQUAD:
    quadPRBS.append(x / quadTOT)
```

this gives

In [35]:
```
print(prbs)
```

```
[0.07908624726793259, 0.016955867977409497, 0.022320950126170365, 0.049695210072
844304, 0.12042206178937831, 0.02036263018930426, 0.022217026016751268, 0.064942
8255016774, 0.0637899174128093, 0.002247358866187966, 0.01019105797991017, 0.040
80645108909219, 0.02417534595361737, 0.068067044041089, 0.07899856130061023, 0.0
1607576067576652, 0.0006300399133532738, 0.05226083652412825, 0.0596784198339162
8, 0.09733142372782276, 0.03033284943669885, 0.008034632709463915, 0.02677344868
909479, 0.0012568321982872007, 0.0228373230448465, 0.0005098776618374432]
```

for the single characters and array sizes for the others as

In [36]:
```
print(len(biPRBS))
print(len(triPRBS))
print(len(quadPRBS))
```

```
676
17576
456976
```

Entropy Estimate

Single Characters

We can now estimate the entropy of the converted text (*all lower case, no special characters, spaces, tabs, or returns*). To do this, we first initialize a variable to hold our value for the entropy

In [37]:
```
entropTOT = 0
```

Then we loop through all the probabilities, computing the entropy for each and adding it to the total

In [38]:
```
for x in prbs:
    entropTOT = entropTOT - x * math.log2(x)
```

to get

In [39]:
```
print(entropTOT)
```

```
4.184820826080936
```

Bi-Grams

For Bi-Grams, we first initialize a variable to hold our value for the entropy

In [40]:
```
biEntropTOT = 0
```

Then we loop through all the Bi-Gram Probabilities

In [41]:
```
testI = 0
for x in biPRBS:
    testI += 1

    if x == 0.0:
        biEntropTOT = biEntropTOT
    else:
        biEntropTOT = biEntropTOT - x * math.log2(x)


print(testI)
print(biEntropTOT)
```

```
676
13.561376307716296
```

Tri-Grams

For Tr-Grams, we first initialize a variable to hold our value for the entropy

In [42]:
```
triEntropTOT = 0
```

Then we loop through all the Tri-Gram Probabilities

In [43]:
```
testI = 0
for x in triPRBS:
    testI += 1

    if x == 0.0:
        triEntropTOT = triEntropTOT
    else:
        triEntropTOT = triEntropTOT - x * math.log2(x)


print(testI)
print(triEntropTOT)
```

```
17576
27.92124606372456
```

Quad-Grams

For Quad-Grams, we first initialize a variable to hold our value for the entropy

In [44]:
```
quadEntropTOT = 0
```

Then we loop through all the Qaud-Gram Probabilities

In [45]:
```
testI = 0
for x in quadPRBS:
    testI += 1

    if x == 0.0:
        quadEntropTOT = quadEntropTOT
    else:
        quadEntropTOT = quadEntropTOT - x * math.log2(x)


print(testI)
print(quadEntropTOT)
```

```
456976
45.63916839392134
```

Times

Overall, it took

In [46]:
```
t1 = time.time()
print(t1-t0)
```

```
103.23156189918518
```