Assignment 4 Question 3, By Amogha Sekhar

PART a)

In [1]:

```
with open('hw4_vocab.txt', 'r') as f:
    vocab = f.read().split('\n')

vocab.remove('')
#print(vocab)
#print(len(vocab))
```

In [2]:

```
with open('hw4_unigram.txt', 'r') as f:
    uni= f.read().split('\n')
uni.remove('')
for i in range(len(uni)):
    uni[i]= int(uni[i])
#print(uni)
#print(len(uni))
```

In [3]:

```
val= sum(uni)
#print(val)
mle_uni= [uni[i]/val for i in range(len(uni))]
#print(mle_uni)
#print(len(mle_uni))
```

ANSWER FOR PART a)

In [4]:

```
#print(mle_uni)

for i in range(len(uni)):
    if vocab[i][0]== 'M' or vocab[i][0]== 'm':
        print(vocab[i], mle_uni[i])
```

```
MILLION 0.002072759168154815
MORE 0.0017088989966186725
MR. 0.0014416083492816956
MOST 0.0007879173033190295
MARKET 0.0007803712804681068
MAY 0.0007298973156289532
M. 0.0007034067394618568
MANY 0.0006967290595970209
MADE 0.0005598610827336895
MUCH 0.0005145971758110562
MAKE 0.0005144626437991272
MONTH 0.00044490959363187093
MONEY 0.00043710673693999306
MONTHS 0.0004057607781605526
MY 0.0004003183467688823
MONDAY 0.00038198530259784006
MAJOR 0.00037089252670515475
MILITARY 0.00035204581485220204
MEMBERS 0.00033606096579846475
MIGHT 0.00027358919153183117
MEETING 0.0002657374141083427
MUST 0.0002665079156312084
ME 0.00026357267173457725
MARCH 0.0002597935452176646
MAN 0.0002528834918776787
MS. 0.0002389900041002911
MINISTER 0.00023977273580605944
MAKING 0.00021170446604452378
MOVE 0.0002099555498894477
MILES 0.00020596851026319035
```

PART b)

In [5]:

```
import pandas as pd
df= pd.read_csv('hw4_bigram.txt', sep="\t", header= None)
```

In [6]:

```
#df
mle_bi = []
for i,row in df.iterrows():
    mle_bi.append(row[2]/uni[(row[0]-1)])
df['mle_bi']= mle_bi
```

In [7]:

```
#df
```

In [8]:

```
#Find index of THE

for i in range(len(vocab)):
    if vocab[i] == 'THE':
        print (i+1)
```

```
4
```

In [9]:

```
df1= df[df[0]== 4]
df1
df1.sort_values(by= ['mle_bi'], inplace= True)
```

```
//anaconda/lib/python3.5/site-packages/ipykernel/__main__.py:3: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pyd
ata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  app.launch_new_instance()
```

In [10]:

```
df2= df1.tail(10)
df2
```

Out[10]:

	0	1	2	mle_bi
1014	4	23	23752	0.006161
1282	4	308	24239	0.006287
1029	4	39	25641	0.006651
1086	4	103	26230	0.006803
1165	4	184	33435	0.008672
1050	4	61	36439	0.009451
1060	4	73	44949	0.011659
1064	4	79	45186	0.011720
1058	4	70	51556	0.013372
993	4	1	2371132	0.615020

In [11]:

```
for i in df2[1]:
    print(vocab[i-1])
```

```
TWO
SAME
NINETEEN
GOVERNMENT
UNITED
NEW
COMPANY
FIRST
U.
<UNK>
```

ANSWER FOR PART b)

The ten most likely words to follow the word “THE”, along with their numerical bigram probabilities is as follows:

Word	**Numerical Bigram Probabilities**
UNK	0.615020
U.	0.013372
FIRST	0.011720
COMPANY	0.011659
NEW	0.009451
UNITED	0.008672
GOVERNMENT	0.006803
NINETEEN	0.006651
SAME	0.006287
TWO	0.006161

PART c)

In [12]:

```
import string
#sentence= 'The stock market fell by one hundred points last week.'

#Function to make a sentence into a list
def strip_sentence(sentence):
    sentence = sentence.translate(str.maketrans('','',string.punctuation))
    s= sentence.split(' ')
    for i in range(len(s)):
        s[i]= s[i].upper()
    return s

#s= strip_sentence(sentence)
#s
```

In [13]:

```
#Given a word, find its index in vocab
def find_index(s):
    a= -1
    for i in range(len(vocab)):
        if vocab[i] == s:
            a= i+1
            break
    return a
```

In [14]:

```
#Function to find the probability that word b follows word a P(b|a)

def prob(a,b):
    c=-1
    x= find_index(a)
    y= find_index(b)
    for i in range(len(df)):
        if df[0][i]== x:
            if df[1][i]== y:
                c= df['mle_bi'][i]
                break
    return c
```

In [15]:

```
#Function to find the probability of a word a

def prob_uni(a):
    x= find_index(a)
    if x== -1:
        return -1
    return mle_uni[x-1]
```

In [16]:

```
#Function to find log-likelihood for a sentence under a unigram model
import math
def logprob_uni(sentence):
    prob_u= 1
    s= strip_sentence(sentence)
    for i in s:
        if prob_uni(i)== -1:
            print(i,"is not in the list of words")
            return -1
        else:
            prob_u= prob_u* prob_uni(i)
            print(prob_u)
    return math.log(prob_u)
```

In [17]:

```
sentence= 'The stock market fell by one hundred points last week.'
logprob_uni(sentence)
```

```
0.047151941408226795
3.1512507432007626e-05
2.4591455775476525e-08
6.506598614749057e-12
2.7199381389047273e-14
1.633507872142784e-16
6.568619493273902e-19
1.450857645838592e-22
1.6837345533215323e-25
9.636204944731571e-29
```

Out[17]:

```
-64.50944034364878
```

In [18]:

```
def logprob_bi(sentence):

    count=0
    p= 1
    s= strip_sentence(sentence)

    for j in range(len(s)):

        if j == 0:

            if prob('<s>',s[j])== -1:
                print('<s>',s[j],"does not have a CPT")
                return -1

            else:
                p= p * (prob('<s>',s[j]))
                print('<s>', s[j])
                print(p)

        else:

            if prob(s[j-1],s[j])== -1:
                print(s[j-1],s[j],"do not have a CPT")
                count= count+1

            else:
                p= p * (prob(s[j-1], s[j]))
                print(s[j-1], s[j])
                print(p)


    if count!=0:
        return -1
    return math.log(p)


logprob_bi(sentence)
```

```
<s> THE
0.158652633836
THE STOCK
0.000369618508476
STOCK MARKET
3.82030439358e-05
MARKET FELL
8.08282936251e-08
FELL BY
1.74479121397e-09
BY ONE
1.42676753746e-11
ONE HUNDRED
2.98280749969e-12
HUNDRED POINTS
1.81440945749e-15
POINTS LAST

1.03479609148e-17
```

```
LAST WEEK
1.69621541031e-18

Out[18]:

-40.91813213378977
```

ANSWER FOR PART c)

The unigram model yields a value of -64.50944034364878, whereas the bigram model yields a value of -40.91813213378977. Therefore, the bigram model yields a higher log-likelihood value .

PART d)

In [19]:

```
sentence= "The sixteen officials sold fire insurance."
logprob_uni(sentence)
```

```
0.047151941408226795
9.538235390711064e-06
6.366061919286568e-09
1.290342318518533e-12
2.7918383925292573e-16
5.811085501680198e-20

Out[19]:

-44.291934473132606
```

In [20]:

```
logprob_bi(sentence)
```

```
<s> THE
0.158652633836
THE SIXTEEN
3.62540532139e-05
SIXTEEN OFFICIALS do not have a CPT
OFFICIALS SOLD
3.3216716644e-09
SOLD FIRE do not have a CPT
FIRE INSURANCE
1.01390690112e-11

Out[20]:

-1
```

ANSWER FOR PART d)

The pair {Sixteen, Officials}, {Sold, Fire} is not observed in the training corpus. The result of this is that the probability becomes 0 and since log(0) is not defined, we get an undefined value.

PART e)

In [21]:

```
def mix_model(sentence, lambda_val):

    prob_mix= 1
    s= strip_sentence(sentence)
    l= lambda_val

    for i in range(len(s)):
        if i==0:
            prob_mix= prob_mix *((prob_uni(s[i])*l)+ (1-l)*prob('<s>', s[i]))
        else:
            if prob(s[i-1],s[i])!= -1:
                prob_mix= prob_mix * ((prob_uni(s[i])*l)+ (1-l)*prob(s[i-1], s[i]))
            else:
                prob_mix= prob_mix* (prob_uni(s[i]))*l

    return math.log(prob_mix)
```

In [22]:

```
import numpy as np
lam= np.arange(0.01, 1.01, 0.01).tolist()
print(lam)
```

```
[0.01, 0.02, 0.03, 0.04, 0.05, 0.060000000000000005, 0.069999999999999
99, 0.08, 0.09, 0.09999999999999999, 0.11, 0.12, 0.13, 0.14, 0.1500000
0000000002, 0.16, 0.17, 0.18000000000000002, 0.19, 0.2, 0.210000000000
00002, 0.22, 0.23, 0.24000000000000002, 0.25, 0.26, 0.27, 0.28, 0.2900
0000000000004, 0.3, 0.31, 0.32, 0.33, 0.34, 0.35000000000000003, 0.360
00000000000004, 0.37, 0.38, 0.39, 0.4, 0.41000000000000003, 0.42000000
000000004, 0.43, 0.44, 0.45, 0.46, 0.47000000000000003, 0.480000000000
00004, 0.49, 0.5, 0.51, 0.52, 0.53, 0.54, 0.55, 0.56, 0.57000000000000
01, 0.5800000000000001, 0.59, 0.6, 0.61, 0.62, 0.63, 0.64, 0.65, 0.66,
0.67, 0.68, 0.6900000000000001, 0.7000000000000001, 0.710000000000000
1, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78, 0.79, 0.8, 0.81, 0.820000
0000000001, 0.8300000000000001, 0.8400000000000001, 0.85, 0.86, 0.87,
0.88, 0.89, 0.9, 0.91, 0.92, 0.93, 0.9400000000000001, 0.9500000000000
001, 0.9600000000000001, 0.97, 0.98, 0.99, 1.0]
```

In [24]:

```
mixed_prob= []
sentence= "The sixteen officials sold fire insurance."
for i in lam:
    #print(i)
    x= mix_model(sentence, i)
    mixed_prob.append(x)
```

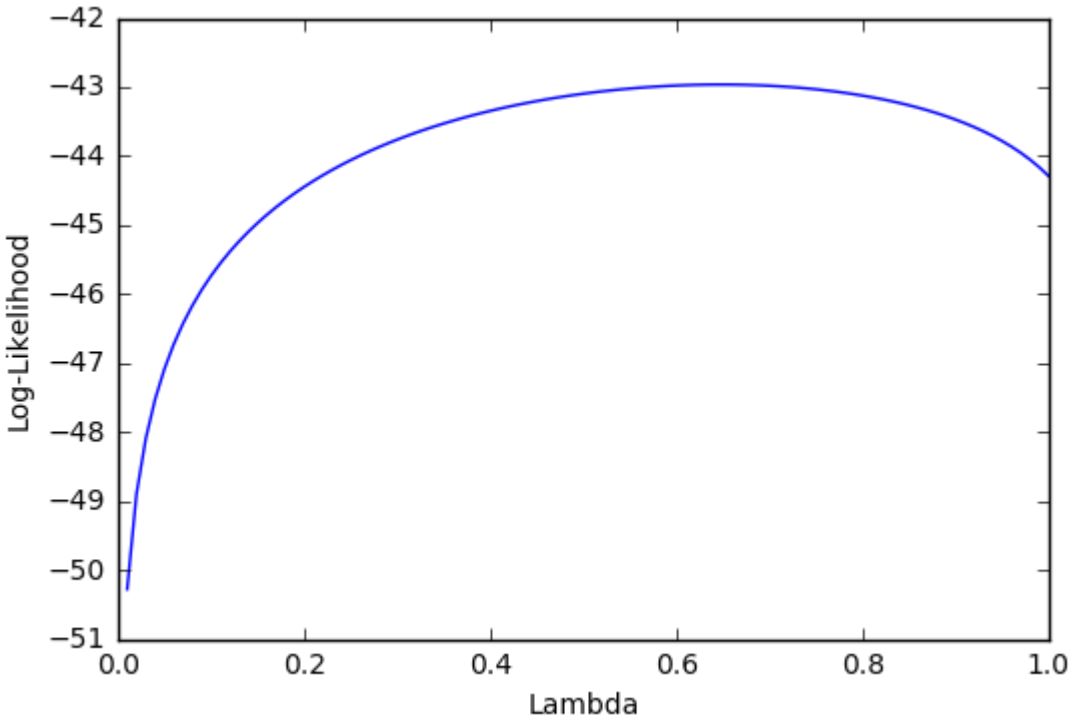
In [25]:

```
%matplotlib inline
import matplotlib.pyplot as plt

plt.plot(lam, mixed_prob)
plt.xlabel('Lambda')
plt.ylabel('Log-Likelihood')
```

Out[25]:

```
<matplotlib.text.Text at 0x11cfb1c88>
```



In [26]:

```
print(max(mixed_prob))
print(mixed_prob.index(max(mixed_prob)))
```

```
-42.96416428296298
64
```

In [27]:

```
lam[64]
```

Out[27]:

```
0.65
```

ANSWER FOR PART e)

ANSWER FOR PART c)

The optimal value of lambda is 0.65.

In [ ]:

```
```