Please ignore the b' at the beginning of words. When reading string from files, numpy.loadtxt() reads as byte strings.

4.3 a)

```
>>> UnigramStartsWith('A')

[WORD, UNIGRAM_PROBABILITY]
["b'A'", '0.018407244690712494'],
["b'AND'", '0.017863233925020615'],
["b'AT'", '0.004312974000612439'],
["b'AS'", '0.003991797167406474'],
["b'AN'", '0.002999256673943544'],
["b'ARE'", '0.002989692670913687 4'],
["b'ABOUT'", '0.0019256178376532746'],
["b'AFTER'", '0.0013465675979453587'],
["b'ALSO'", '0.0013100115812493978'],
["b'ALL'", '0.0011818148040640 31'],
["b'A.'", '0.0010256109080316418'],
["b'ANY'", '0.0006318601694814718'],
["b'AMERICAN'", '0.0006120961939108219'],
["b'AGAINST'", '0.0005959645826622 53'],
["b'ANOTHER'", '0.0004283866165304179'],
["b'AMONG'", '0.00037429251755208585'],
["b'AGO'", '0.0003565709825261751'],
["b'ACCORDING'", '0.0003475451075440342'],
["b'AIR'", '0.0003110013210309760 4'],
["b'ADMINISTRATION'", '0.0002915186396670866'],
["b'AGENCY'", '0.0002796553622515356'],
["b'AROUND'", '0.0002768546503668333 5'],
["b'AGREEMENT'", '0.0002627899400288089 5'],
["b'AVERAGE'", '0.0002590719644264094 3'],
["b'ASKED'", '0.0002582280818061279 5'],
["b'ALREADY'", '0.0002490799049949608'],
["b'AREA'", '0.0002310893059451922'],
["b'ANALYSTS'", '0.0002260382404064060 4'],
["b'ANNOUNCED'", '0.0002271511870505453 6'],
["b'ADDED'", '0.0002212195483427698 6'],
["b'ALTHOUGH'", '0.0002142605742711734 5'],
["b'AGREED'", '0.0002117778471419395 7'],
["b'APRIL'", '0.0002066900910544455 2'],
["b'AWAY'", '0.0002020548517343487 8']
```

4.3 b)

```
>>> BigramNext('THE', 5)

["b'<UNK>'", '0.6150198100055118'],
["b'U.'", '0.013372499432610317'],
["b'FIRST'", '0.011720260675031612'],
["b'COMPANY'", '0.011658788055636611'],
["b'NEW'", '0.009451480076516552']
```

4.3 c)

```
>>> UnigramSentence('last week the stock market fell by one hundred
points')

-64.50944034364878

>>> BigramSentence('last week the stock market fell by one hundred
points')

-44.74046921340373
```

The Bigram model yields the higher Loglikelihood for this sentence compared to Unigram model.

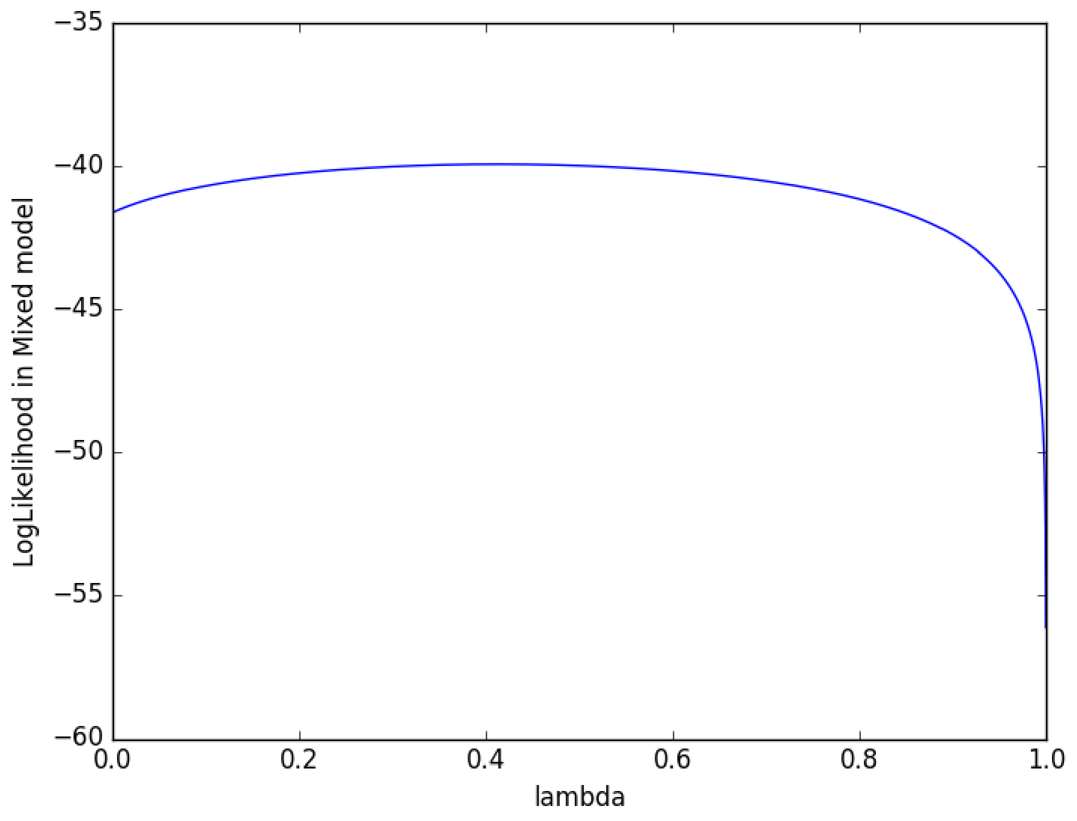
4.3 d)

```
>>> UnigramSentence('the nineteen officials sold fire insurance')

-41.64345971649364

>>> BigramSentence('the nineteen officials sold fire insurance')

b'NINETEEN' b'OFFICIALS'
b'SOLD' b'FIRE'
-inf
```

In the bigram model, the two pairs listed above are not observed together in the corpus. Therefore, the likelihood of them appearing together in a sentence is 0, hence loglikelihood of such a sentence tends to -inf (log 0).

4.4 e)

```
>>> lmbdFunc ('the nineteen officials sold fire insurance', .0001)

Optimum lambda: 0.4126
Maximum Likelihood: -39.9536375037
```



4.4 e)

```
"""
Created on Sun Oct 23 00:32:51 2016
@author: gopal
"""

import numpy as np;
import math;
import matplotlib.pyplot as plt;

vocab = np.loadtxt('vocab250A.txt', dtype=str, unpack=True)
vocabCount = np.loadtxt('unigram.txt', dtype=float, unpack=True)
preWordId, nextWordId, followCount = np.loadtxt('bigram.txt', dtype=int, unpack=True)

#returns index of a word in vocab. fixes the byte encoding b' match.
def WhatIndex(s):
    s = "b'" + s + "'"
    result = np.where(vocab==s)[0]
    if len(result) == 0 :
        return -1
    else:
        return result[0]

lenVocab = np.size(vocab);
totalWords = np.sum(vocabCount)
unigramP = np.divide(vocabCount, totalWords)

bigramP = np.zeros(shape=(lenVocab, lenVocab), dtype=float)

for i in range(len(preWordId)):
    bigramP[preWordId[i]-1][nextWordId[i]-1] = followCount[i]
bigramP = bigramP/bigramP.sum(axis=1, keepdims=True)

def UnigramStartsWith(s):
    s = s.upper()
    result1Index = np.array([i for i in range(lenVocab) if vocab[i][2]==s])
    result = np.column_stack((vocab[result1Index], unigramP[result1Index]))
    return result

def BigramNext(prevWord, howMany):
    prevWord = prevWord.upper()
    prevIndex = WhatIndex(prevWord)
    if prevIndex != -1 :
        nextWord = bigramP[WhatIndex(prevWord)]
        result2Index = np.argsort(nextWord)[::-1]
        result = np.column_stack((vocab[result2Index][:howMany],
nextWord[result2Index][:howMany]))
        return result
    else:
        return "Word not found!"

def UnigramSentence(s):
    s =s.upper();
    logLikelihood = 0.0
    for w in s.split():
        indexS = WhatIndex(w)
        if indexS != -1:
            logLikelihood += math.log(unigramP[indexS])
        else:
            logLikelihood -= math.inf
            print (w)
    return logLikelihood;

def BigramSentence(s):
    s =s.upper();
    logLikelihood = 0.0
    s = '<s> ' + s
    splitS = s.split();
    indices = np.array([WhatIndex(w) for w in splitS])
    for i in range(1,len(splitS)):
        if indices[i]!=-1 and indices[i-1]!= -1:
            if bigramP[indices[i-1]][indices[i]] > 0.0:
                logLikelihood += math.log(bigramP[indices[i-1]][indices[i]])
            else:
                logLikelihood -=math.inf
                print (vocab[indices[i-1]], vocab[indices[i]])
        else :
            logLikelihood -=math.inf
            print (vocab[indices[i-1]], vocab[indices[i]])
    return logLikelihood
```

```
def MixedSentence(s, lmda):
    s = s.upper()
    s = '<s> ' + s
    logLikelihood = 0.0
    splitS = s.split();
    indices = np.array([WhatIndex(w) for w in splitS])
    for i in range(1,len(splitS)):
        if indices[i]!=-1 and indices[i-1]!= -1:
            Pb = bigramP[indices[i-1]][indices[i]]
            Pu = unigramP[indices[i]]
            mixedP = (1-lmda)*Pu + lmda*Pb
            if mixedP > 0.0:
                logLikelihood += math.log(mixedP)
            else:
                logLikelihood -= math.inf
        else:
            logLikelihood -=math.inf
            print (vocab[indices[i-1]], vocab[indices[i]])
    return logLikelihood

def lmbdFunc(s, step):
    xRange = np.arange(0,1,step)
    y = np.array([MixedSentence(s, x) for x in xRange])
    print ('Optimum lambda:', xRange[np.argmax(y)])
    print ('Maximum Likelihood: ' , np.max(y))
    plt.plot(xRange, y)
    plt.xlabel('lambda')
    plt.ylabel('LogLikelihood in Mixed model')
```