CS6120 Assignment 1

October 11, 2018

1 Naive Bayes Document Classification

We must compute several values,

Priors:

$P(c) = \frac{N_c}{N}$ where N_c is just number of documents with class and N number of documents

We will calculate the conditional probabilities of each word in the document. For the purposes of this calculation we will not calculate conditional probabilities for every single word, but only the words in D1 and D2

Using

$$P(w|c) = \frac{count(w,c) + \lambda}{count(c) + |V| \cdot \lambda}$$

Using $\lambda = 0.1$ Example calculation:

$P(rose|vegetable) = \frac{0+0.1}{8+7 \cdot 0.1}$ Other calculations outlined below

We then find the maximum probablity of a document being in a class by using Where c is class and d document $P(c|d) = P(c) \cdot \prod_i^n P(d_i|c)$

Example calculation: $P(flower|D1) = P(flower) \cdot P(rose|flower) \cdot P(lily|flower) \cdot P(apple|flower) \cdot P(carrot|flower)$

```
In [1]: def p(wc, c, v, l=0.1):
            return (wc + l)/(c + v * l)

        P={}

        P[('rose', 'vegetable')] = p(0, 8, 7)
        P[('lily', 'vegetable')] = p(0, 8, 2)
        P[('apple', 'vegetable')] = p(0, 8, 2)
        P[('carrot', 'vegetable')] = p(1, 8, 2)

        P[('rose', 'flower')] = p(6, 13, 7)
        P[('lily', 'flower')] = p(1, 13, 2)
        P[('apple', 'flower')] = p(0, 13, 2)
        P[('carrot', 'flower')] = p(0, 13, 2)

        P[('rose', 'fruit')] = p(1, 14, 7)
        P[('lily', 'fruit')] = p(1, 14, 2)
        P[('apple', 'fruit')] = p(2, 14, 2)
```

```
P[('carrot', 'fruit')] = p(1, 14, 2)

#Priors
P['vegetable'] = 1/4
P['flower'] = 3/8
P['fruit'] = 3/8

D1_flower = P['flower']*P[('rose', 'flower')]*P[('lily', 'flower')]*P[('apple', 'flower'
print("D1_flower", D1_flower)
D1_fruit = P['fruit']*P[('rose', 'fruit')]*P[('lily', 'fruit')]*P[('apple', 'fruit')]*P[
print("D1_fruit", D1_fruit)
D1_vegetable = P['vegetable']*P[('rose', 'vegetable')]*P[('lily', 'vegetable')]*P[('appl
print("D1_vegetable", D1_vegetable)
```

```
D1_flower 7.985671244629444e-07
D1_fruit 2.490268929586247e-05
D1_vegetable 5.732867232465228e-08
```

We take the argmax of these values and find that the fruit class is the most probable. Similarly for D2

```
In [2]: P[('pea', 'vegetable')] = p(2, 8, 3)
        P[('lotus', 'vegetable')] = p(1, 8, 2)
        P[('grape', 'vegetable')] = p(0, 8, 2)

        P[('pea', 'flower')] = p(1, 13, 3)
        P[('lotus', 'flower')] = p(0, 13, 2)
        P[('grape', 'flower')] = p(0, 13, 2)

        P[('pea', 'fruit')] = p(0, 14, 3)
        P[('lotus', 'fruit')] = p(1, 14, 2)
        P[('grape', 'fruit')] = p(2, 14, 2)

        D2_flower = P['flower']*(P[('pea', 'flower')]**2)*P[('lotus', 'flower')]*P[('grape', 'fl
        print("D2_flower", D2_flower)
        D2_fruit = P['fruit']*(P[('pea', 'fruit')]**2)*P[('lotus', 'fruit')]*P[('grape', 'fruit'
        print("D2_fruit", D2_fruit)
        D2_vegetable = P['vegetable']*(P[('pea', 'vegetable')]**2)*P[('lotus', 'vegetable')]*P[(
        print("D2_vegetable", D2_vegetable)
```

```
D2_flower 1.47219552641001e-07
D2_fruit 2.1008472857159783e-07
D2_vegetable 2.618107011591733e-05
```

We find that D2 is classed as vegetable

2 Word Sense Disambiguation

Counting all the senses will be done by putting each word through wordnet

In the cold weather, they started to the city. They were least worried protecting themselves against the common cold. After she signed the agreement, a cold chill crept up her spine. “Chill, its not that serious,” her husband assured and left to deposit cash at the bank.

```
In [3]: from nltk import download
        download('wordnet')
        download('punkt')

[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\cdilg\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\cdilg\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!


Out[3]: True

In [228]: from nltk.corpus import wordnet as wn
          import numpy as np
          import string

          raw = "In the cold weather, they started to the city. They were least worried protecti
          sents = [s.translate(str.maketrans('','', string.punctuation)).lower() for s in raw.st

          sentence_senses = []
          word_senses = {}
          for s in sents:
              sentencecount = 1
              for word in s.split(' '):
                  syns = max([len(wn.synsets(word)), 1])
                  sentencecount *= syns
                  word_senses[word] = syns
              sentence_senses += [sentencecount]

          total = 1
          for sentence in sentence_senses:
              total *= sentence
          print("Total senses: ", total)
          print("Distinct combinations of senses per sentence: ", sentence_senses)
          #print(word_senses)

Total senses:  4654630885130005118976000
Distinct combinations of senses per sentence:  [28224, 149760, 39513600, 27869184]
```

Here we see how many different ways all of the senses could be combined, broken down by sentence. These numbers are extremely large for a relatively short sentence of only around 10 words.

Language Modelling

Implement a 4 gram language model

In [77]:
```
from os import listdir
from nltk import word_tokenize
import pickle
from collections import Counter

def save(corpus, file):
    with open(file, 'wb') as f:
        pickle.dump(corpus, f, pickle.HIGHEST_PROTOCOL)

def read(file):
    with open(file, 'rb') as f:
        return pickle.load(f)

def read_dir(directory, cachefile):
    try:
        text = read(cachefile)
    except(FileNotFoundError):
        corpus = ""
        base = directory
        for file in listdir(base):
            for line in open(base + "/" + file):
                corpus += ' ' + line.strip().lower().replace('  ', ' ')
        text = word_tokenize(corpus)
        save(text, cachefile)
    finally:
        return text

# we need to remove words that occur less than 5 times and replace with UNK
# count the items in the list. Figure out which ones are greater
# unkwords = [w for w in [w for w in wordcount.keys() if wordcount[w] <= unk_threshold]

# we want a list of indices for which to replace with 'UNK'
# go through the list, keep an index of where each word ocurrs.
# at the end, count all of the lengths of these lists
# for each list which is less than 5. go to the text list and replace each element with

def replace_unk(text, threshold, savefile):
    try:
        return read(savefile)
    except(FileNotFoundError):
        counterdict = {}
        for i, t in enumerate(text):
```

```
            if t in counterdict.keys():
                counterdict[t].append(i)
            else:
                counterdict[t] = [i]

        for locations in counterdict:
            #print(locations, len(counterdict[locations]))
            if len(counterdict[locations]) <= threshold:
                for loc in counterdict[locations]:
                    text[loc] = 'UNK'
        save(text, savefile)
        return text

#find out the definition of 4 gram counts
#probably count all of the ways 3 previous words occur
#make a big table

def ngram(n, text, outfile):
    ngrams = {}
    for i in range(n, len(text)+1):
        #get the previous n words.
        gram = tuple(text[i-n:i])
        if gram in ngrams.keys():
            ngrams[gram] += 1
        else:
             ngrams[gram] = 1
    #save a textual representation of the dict to file

    with open(outfile, 'w') as f:
        for line in sorted(ngrams, key=ngrams.get, reverse=True):
            f.write(' '.join(line) + ' ' + str(ngrams[line]) + '\n')
    return ngrams
```

In [78]:
```
text = read_dir('gutenberg', 'gutenberg-corpus.txt')
text = replace_unk(text, 5, 'gutenberg-unk.txt')
guten4 = ngram(4, text, 'gutenberg-4grams.txt')
guten3 = ngram(3, text, 'gutenberg-3grams.txt')
```

Perplexity formula: $PP(W) = \left(\prod_{i=1}^{N} \frac{1}{P(w|w_{n-1}, w_{n-2}, w_{n-3}, w_{n-4})}\right)^{\frac{1}{N}}$

Probability of words: $P(w) = \frac{numcounts}{totalwords}$ $P(w_n|w_{n-1}, w_{n-2}, w_{n-3}, w_{n-4}) = \frac{C(w_{n-4}w_{n-3}w_{n-2}w_{n-1}w)+0.1}{C(w_{n-4}w_{n-3}w_{n-2}w_{n-1})+|V|\times 0.1}$

In [79]:
```
imdb = read_dir('imdb_data', 'imdb-corpus.txt')
imdb = replace_unk(imdb, 5, 'imdb-unk.txt')
#imdbmodel = ngram(4, imdb, 'imdb-ngrams.txt')
#imdbmodel3 = ngram(3, imdb, 'imdb-3grams.txt')
import math
```

```
def calculate_probability(model, onelessmodel, word, context, n = 4, lam = 0.1):
    #for each of the words, we need the prior 4 words. We will look this up and find wh
    try:
        ret = (model[(context[0], context[1], context[2], word)] + lam)/(onelessmodel[(
    except:
        ret = lam/(len(model)*lam)
    #here v is a vocabulary of n-grams, so will be the count of the ngrams

    return ret


def perplexity(model, onelessmodel, text, n = 4):
    #this takes in text, which the existing probabilities and counts are used to
    #come up with a number, all of the probabilities multiplied together. We probably c
    #Perplexity is a measure of how probable the model is at generating a sentence
    #
    #Perplexity is an integer - lower better
    pp = 1
    V = len(model)
    for i, word in enumerate(text):
        #calculate the probability of this word
        #TODO implement log sum instead
        pp *= 1/calculate_probability(model, onelessmodel, word, text[i-n:i-1])
    return math.pow(pp, 1/V)

def wikiperplexity(model, onelessmodel, text, n=4):
    pp = 0
    V = len(model)
    for i, word in enumerate(text):
        pp -= math.log(2, calculate_probability(model, onelessmodel, word, text[i-n:i-1
    return math.pow(2, pp/V)
#Currently, this works:
#print(guten4[('the', 'children', 'of', 'israel')])

print(calculate_probability(guten4, guten3, 'israel', ['the', 'children', 'of']))
print(calculate_probability(guten4, guten3, 'were', ['children', 'of', 'israel']))
print(wikiperplexity(guten4, guten3, ['the', 'children', 'of', 'israel', 'were', 'off',
print(perplexity(guten4, guten3, ['the', 'children', 'of', 'israel', 'were', 'off', 'on
#perplexity(guten4, guten3, ["the", "children", "of", "israel", "are", "well"])
```

```
0.0030798517076259754
7.797958291094393e-05
1.0000001605774118
1.000070634697219
```

```
In [80]: news = read_dir('news_data', 'news-corpus.txt')
         news = replace_unk(news, 5, 'news-unk.txt')
```

```
news4 = ngram(4, news, 'news-4grams.txt')
news3 = ngram(3, news, 'news-3grams.txt')

print(wikiperplexity(news4, news3, ['the', 'children', 'of', 'israel', 'were', 'off', '
print(perplexity(news4, news3, ['the', 'children', 'of', 'israel', 'were', 'off', 'on',
```

```
1.0000033056111426
1.0009467110114756
```

2.1 POS Tagging HMM

First find the tag unigram and tag bigram counts from the corpus

In [115]:
```
import operator
import random
#read in the file/s?
import nltk
from nltk.corpus import brown

#nltk.download('brown')
def read_brown(directory, cachefile):
    try:
        text = read(cachefile)
    except(FileNotFoundError):
        corpus = []
        base = directory
        for file in listdir(base):
            #print(file)
            for sent in open(base + "/" + file):
                if sent == "\n": continue
                wordlist = []
                for word in sent.strip().split(' '):
                    #split the word and it's tag
                    if word == '':
                        continue
                    wordlist.append(word.split('/'))

                corpus += [['<s>', '<s>']] + wordlist + [['</s>', '</s>']]
        text = corpus
        save(text, cachefile)
    finally:
        return text
brown = read_brown('brown', 'brown-cache.txt')

#calculate the word-tag counts
#lets do this in the same dictionary way we did earlier
```

```python
def wordtag(text, outfile):
    pairs = {}
    for word in text:
        #ignore the sentence tags
        if (word == ['<s>', '<s>'] or word == ['</s>', '</s>']): continue
        tagpair = tuple(word)
        if tagpair in pairs.keys():
            pairs[tagpair] += 1
        else:
             pairs[tagpair] = 1
    #save a textual representation of the dict to file

    with open(outfile, 'w') as f:
        for word in sorted(pairs, key=pairs.get, reverse=True):
            f.write(' '.join(word) + ' ' + str(pairs[word]) + '\n')
    return pairs

def tagunigram(text, outfile):
    '''This is literally a unigram of the tag, t_n. That is we
    will not consider the word association and will instead just
    consider the impact of the counts of tags themselves.'''
    unigrams = {}
    for word in text:
        #ignore the sentence tags
        if (word == ['<s>', '<s>'] or word == ['</s>', '</s>']): continue
        tag = word[1]
        if tag in unigrams.keys():
            unigrams[tag] += 1
        else:
             unigrams[tag] = 1
    #save a textual representation of the dict to file

    with open(outfile, 'w') as f:
        #TODO there is a problem with the way this joins - it's assuming a tuple
        for word in sorted(unigrams, key=unigrams.get, reverse=True):
            f.write(' '.join(word) + ' ' + str(unigrams[word]) + '\n')
    return unigrams

def savecounts(d, file):
    with open(file, 'w') as f:
        for token in sorted(d, key=d.get, reverse=True):
            f.write(' '.join(token) + ' ' + str(d[token]) + '\n')

def tagbigram(text, outfile):
    '''Here we consider both t_n and t_n-1 and report the counts.
    Again we do not stop to consider the effects of the word association'''
    bigrams = {}
    for i in range(len(text)):
```

```
            #ignore the sentence tags
            if (text[i] == ['<s>', '<s>'] or text[i] == ['</s>', '</s>']): continue
            t = text[i][1]
            t1 = text[i-1][1]
            if (t, t1) in bigrams.keys():
                bigrams[(t, t1)] += 1
            else:
                 bigrams[(t, t1)] = 1
        savecounts(bigrams, outfile)
        return bigrams
        #save a textual representation of the dict to file


    def transition(bigramtags, unigramtags):
        probabilities = {}
        for bigram in bigramtags.keys():
            probabilities[bigram] = bigramtags[bigram]/unigramtags[bigram[0]]
        savecounts(probabilities, 'brownmeta/transition-probabilities.txt')
        return probabilities

    def emission(wordtags, unigramtags):
        emissionprob = {}
        for wordpair in wordtags.keys():
            emissionprob[wordpair] = wordtags[wordpair]/unigramtags[wordpair[1]]
        savecounts(emissionprob, 'brownmeta/emission-probabilities.txt')
        return emissionprob



    tags = wordtag(brown, 'brownmeta/brownwordtag.txt')
    unitags = tagunigram(brown, 'brownmeta/brownuni.txt')
    bitags = tagbigram(brown, 'brownmeta/brownbigrams.txt')

    #These are both saved to file
    transitionprobs = transition(bitags, unitags)
    emissionprobs = emission(tags, unitags)
```

In [187]:
```
from collections import defaultdict
class postagger:

    def default(self):
        return 0.0001

    def __init__(self, wordtags, unigramprobabilities, bigramprobabilities):
        #Emission probabilities
        self.wt = defaultdict(self.default, wordtags)

        self.up = defaultdict(self.default, unigramprobabilities)
```

```
        #Transition probabilities
        self.bp = defaultdict(self.default, bigramprobabilities)

    def nextword(self, dct):
        #select a next item based on a random number which is weighted by the probabil

        #sum all of the probabilities and normalize
        tot = 0
        for value in dct.keys():
            tot += dct[value]
        normalised = {}
        index = random.random()
        for pair in dct.keys():
            index -= dct[pair] / tot
            if index <= 0.0:
                return pair

    def predictSentence(self):
        '''Will generate a sentence, with associated tags.
        Output will contain sentence and sentence probability in a dict'''
        sent = []
        humansent = []
        priortag = '<s>'
        sentp = 0
        while(priortag != '</s>' and priortag != '.'):
            subset = {}
            for tags in self.bp.keys():
                if priortag == tags[1]:
                    subset[tags] = self.bp[tags]
            selectedbigram = self.nextword(subset)

            #capture tag probability
            sentp -= math.log(subset[selectedbigram], 2)
            currenttag = selectedbigram[0]

            potentialwordtags = {}

            for wordtag in self.wt.keys():
                if currenttag == wordtag[1]:
                    potentialwordtags[wordtag] = self.wt[wordtag]
            currentword = self.nextword(potentialwordtags)
            #capture word probability
            sentp -= math.log(potentialwordtags[currentword], 2)

            sent.append('/'.join(currentword))
            #humansent.append(currentword[0])
            priortag = currenttag
```

```
        return({'sentence': sent, 'probability': math.pow(2, sentp)})

    def viterbi(self, sentence):
        '''Takes a tokenised sentence and will then apply some tags to it.'''
        #remember states are the wordtags
        startp = {}
        for tags in self.bp.keys():
                if '<s>' == tags[1]:
                    startp[tags[0]] = self.bp[tags]

        viterbim = [{}]
        #we will keep track of the backpointers using a list of dicts, with probabilit
        #this will make the backtracing easy

        #Essentially the up.tags gives us a list of all of the POS tags
        for state in startp.keys():
            #Create the first column of the viterbi

            #TODO implement lambda smoothing. It will require changing how the probabi
            #and will require a return here for the unknowns which is calculated in pl

            viterbim[0][state] = {'prev': None, 'probability': startp[state]*self.wt[(

        for i in range(1, len(sentence)):
            #Find the maximum transition probability from the previous state to the cu
            viterbim.append({})
            for state in self.up.keys():
                viterbim[i][state] = {'probability': 0, 'prev': None}
                listofstateprobs = []
                for prevstate in viterbim[i-1].keys():
                    #TODO Lambda smoothing
                    currentprob = viterbim[i-1][prevstate]['probability']*self.bp[(pre
                    currentmax = viterbim[i][state]['probability']
                    if (currentprob > currentmax):
                        #This is basically saying, set the probability to the probabil
                        #and multiply (by markov assumption) the emission probability

                        viterbim[i][state] = {'probability': currentprob*self.wt[(sent
        #now find the highest probability state
        taggedsentence = []
        maxprob = max(prob['probability'] for prob in viterbim[-1].values())
        #backtrack on this state
        prevstate = None

        #iterate through backwards through viterbim

        for state, prob in viterbim[i].items():
                if prob['probability'] == maxprob:
```

```
                        prevstate = prob['prev']
                        taggedsentence.append('/'.join([sentence[i], state]))

                for i in range(len(viterbim)-1, 0, -1):
                    prevstate = viterbim[i][prevstate]['prev']
                    taggedsentence.insert(0, '/'.join([sentence[i-1], prevstate]))
                return taggedsentence


        pos = postagger(tags, unitags, bitags)
        sents = []
        with open('brownmeta/generatedsentences.txt', 'w') as f:
            for i in range(5):
                sents.append(pos.predictSentence())
                f.write('{} Probability: {}\n'.format(' '.join(sents[i]['sentence']), sents[i]

        with open('brownmeta/human-readablesentences.txt', 'w') as f:
            for sent in sents:
                f.write(' '.join([word.split('/')[0] for word in sent['sentence']]) + '\n')

        pos.viterbi(['The', 'cat', 'sat'])
```

```
maxprob: 79410481308774000
3
```

```
Out[187]: ['The/at', 'cat/nn-nc', 'sat/vbd']
```

```
In [214]: import re
          with open('brownmeta/science_sample.txt') as f:
              sentencecounter = 0
              sentence = False

              words = [[]]
              for line in f:
                  if line != '\n':
                      if re.match('<.*>', line):
                          if sentence:
                              words.append([])
                              sentencecounter += 1
                          sentence = not sentence
                      else:
                          words[sentencecounter].append(line.strip())
          tags = []
          for sentence in words:
              try:
                  tags.append(pos.viterbi(sentence))
              except:
```

```python
                tags.append(['error'])
        with open('brownmeta/science-tagged.txt', 'w') as f:
            for sent in tags:
                f.write(' '.join(sent) + '\n')
```

```
maxprob: 2.0080252384283956e+42
8
maxprob: 3.1477207761221576e+97
20
maxprob: 3.021324982380108e+109
24
maxprob: 2.1955339299622646e+239
53
maxprob: 3.7176547643644863e+59
13
maxprob: 3.365011980748336e+52
12
maxprob: 9.942807467315857e+33
10
maxprob: 1.5800078389366676e+245
58
maxprob: 7.275540136710308e+34
9
maxprob: 4.654784038470841e+39
9
maxprob: 3.310088804696931e+100
25
maxprob: 6.820221933502801e+34
9
maxprob: 1.3794921079771193e+108
23
maxprob: 2.740226758248556e+25
8
maxprob: 2.3085513786579857e+85
19
maxprob: 7.977017574726705e+25
7
maxprob: 3.183289012453947e+49
13
maxprob: 1.5337830574162862e+38
9
maxprob: 1.8049563962042477e+107
20
maxprob: 3.748864718831112e+110
23
maxprob: 8.428131337233712e+59
12
```

```
maxprob: 1.3450249446538133e+109
25
maxprob: 4.28122840840757e+201
44
maxprob: 6.49513912383942e+61
15
maxprob: 1.5928714459221863e+107
19
maxprob: 5.729089095576724e+143
30
maxprob: 1.510424638859083e+67
16
maxprob: 1.8316315259107136e+31
7
maxprob: 3.191219054616047e+51
11
maxprob: 2.26263753523917e+145
28
maxprob: 2.960978726747843e+57
15
maxprob: 7.746246496695334e+46
12
maxprob: 2.3341782509392293e+113
23
maxprob: 3.713256797713509e+43
11
maxprob: 5.519938228928071e+54
10
maxprob: 1.7046322881813378e+16
5
maxprob: 3.6537752263632495e+43
10
maxprob: 3.954091780817073e+44
10
maxprob: 4.31954834752815e+84
17
maxprob: 2.8280124839255855e+133
29
maxprob: 5.3407547374234453e+154
31
maxprob: 3.380948357661286e+107
21
maxprob: 3.581773588317232e+44
10
maxprob: 1.305793203418419e+116
26
maxprob: 1.3771973742082674e+36
8
```

```
maxprob: 1.4775691065458468e+28
8
maxprob: 9.440871938310182e+110
24
maxprob: 5.617373723191542e+144
28
maxprob: 7.741484732284823e+93
19
maxprob: 4.001593671241936e+107
24
maxprob: 1.7360063135753248e+89
22
maxprob: 212661603.8753472
4
maxprob: 4.931295792322579e+90
23
maxprob: 1.6368997960539327e+97
21
maxprob: 1.7204832369954207e+129
27
maxprob: 6.2979961160194436e+134
28
maxprob: 2.8544055782657124e+126
26
maxprob: 9.823541967650845e+136
28
maxprob: 5.697256678785799e+193
44
maxprob: 1.0446060398764055e+131
26
maxprob: 2.175320936354649e+160
36
maxprob: 7.695136981167532e+120
27
maxprob: 9.921333325221339e+104
22
maxprob: 3.0017683212328973e+61
12
maxprob: 1.0838893083876078e+161
34
maxprob: 2.867499351537756e+121
27
maxprob: 3.2040299577198434e+130
32
maxprob: 8.954832599396884e+79
18
maxprob: 4.336372238683612e+52
11
```

```
maxprob: 3.251500483195947e+179
38
maxprob: 5.674241268504574e+119
24
maxprob: 1.775344888957056e+53
14
maxprob: 5.502641123237151e+152
31
maxprob: 1.883915522329825e+51
12
maxprob: 2.4678066890219872e+94
23
maxprob: 3.0105336171173067e+128
27
maxprob: 3.701778982837172e+125
25
maxprob: 1.0976713275912227e+74
19
maxprob: 1604918.259
2
maxprob: 8.975327625602733e+29
9
maxprob: 5.26543346877114e+44
10
maxprob: 1.574792699541144e+55
14
maxprob: 1107886.8360000001
2
maxprob: 8.02678948239168e+116
26
maxprob: 2.51938296116893e+68
13
maxprob: 2.326874629022499e+55
13
maxprob: 2.5357280929575984e+61
15
maxprob: 1.599239186634182e+79
17
maxprob: 2.2500717676208848e+95
23
maxprob: 3.963044357286545e+33
8
maxprob: 3.171678480247068e+77
16
maxprob: 9.444038584823708e+38
9
maxprob: 2.1360312867866232e+60
13
```

```
maxprob: 4.487991509940431e+149
30
maxprob: 3.8800517533478456e+101
23
maxprob: 9.148845091120974e+54
11
maxprob: 2.72577046913966e+90
22
maxprob: 3.164706296162717e+129
26
maxprob: 4.2589190093272573e+155
33
maxprob: 9.234051881128154e+103
25
maxprob: 1.039126474489869e+66
15
maxprob: 4.772271343840981e+71
17
maxprob: 1.897406793038261e+46
13
maxprob: 3.2126857694855814e+51
11
maxprob: 2.3967301677703837e+101
21
maxprob: 5.1337339554480445e+72
19
maxprob: 1.5444110521454043e+91
18
maxprob: 2.865100930704463e+60
12
maxprob: 1.1409053959390534e+96
22
maxprob: 3.068452702552526e+43
10
maxprob: 4.067703096288416e+70
13
maxprob: 4.84859722293412e+23
6
maxprob: 3.013171453407726e+99
22
maxprob: 4.364355347705655e+93
22
maxprob: 1.1893513596223013e+49
10
maxprob: 1.1676331294179756e+142
30
maxprob: 2.0451236070539533e+100
27
```

```
maxprob: 6.715470073878611e+20
5
maxprob: 5.404497491073148e+114
24
maxprob: 1.0974788953983454e+139
29
maxprob: 1.170530852206504e+69
15
maxprob: 1.6357992787059615e+145
33
maxprob: 1.315581953515767e+89
21
maxprob: 1.2413454568060485e+48
10
maxprob: 1.2996764365558676e+132
26
```

2.2 Discussion of Viterbi HMM POS Tagger

It is acknowledged that the smoothing function employed in this implementation of the veterbi algorithm is incorrect, and simply adds a constant to all unknown values. This will have a definite negative impact on performance, where unknown bigrams will have higher probability than infrequent but known bigrams.

Look in the folder brownmeta/science-tagged.txt for the sentences and their associated tags.