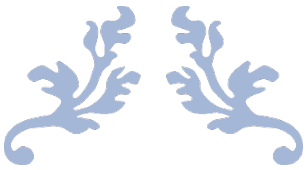

CIS 668 – NATURAL LANGUAGE PROCESSING

Assignment 1

SEPTEMBER 26, 2017

ABHIRAM SRINIVASAN
SUID - 407650467

1. a) I chose the two books from the Gutenberg collection on the web(http://www.gutenberg.org/wiki/Category:Bookshelf).
   To yield a variety of questions I chose two topics of my interest which are of different genres. Since I interned at Boeing the interest towards aviation attracted me to choose “The Early History of the Airplane by Orville Wright and Wilbur Wright” from the Technology genre. Secondly, the rich culture in India made me choose “Modern India by Eleroy Curtis” from the history genre.
   Steps I followed to download and read these books
   - I downloaded both the books and saved in a “.txt” format into the folder from which I will carry out my assignment
   - I used import os and then typed os.getcwd to get the current working directory
   - Once I could access that directory I opened and saved my raw text locally

Loading Flight book

In [64]:
```
import nltk
import os
```

In [65]:
```
os.getcwd()
```

Out[65]:
```
'/Users/abhiram/NLP_Assignment'
```

In [66]:
```
#read the flight book file

flight = open('Flight.txt')
text = flight.read()
```

Loading Indian book

```
In [1]: import nltk
        import os
```

```
In [2]: os.getcwd()
```

```
Out[2]: '/Users/abhiram/NLP_Assignment'
```

```
In [3]: # read the Modern India book file

        india = open('Modern India.txt')
        text = india.read()
```

The lengths of both the files are huge.

Length for Fight book

```
In [110]: # printing the length of the text
          # before tokenization

          print(len(text))
```

```
101556
```

Length for Indian book

```
In [32]: # printing the length of the text
         # before tokenization

         print(len(text))
```

```
864466
```

2.a) Processing options I covered

For Flight book

- Tokenization – I performed tokenization because this book consisted of many unnecessary characters and punctuations. Also, with tokenization we can chop of each word into smaller units.

```
In [70]: # performing tokenization on the Flight text

         flightTokens = nltk.word_tokenize(text)
```

- Lowercase – I converted the tokenized text into lower case to make all the words accept to a universal case format. This would ensure that the system is not in a dilemma when it considers “The” and “the”. I also printed the length of the tokenized words after converting them to lower case.

```
In [73]: # convert the tokenized words to lower case

         flightwords = [w.lower() for w in flightTokens]
```

```
In [71]: # print the length of tokenized words

         print(len(flightwords))
```

```
19430
```

- Frequency of top 50 words – By using from nltk import FreqDist we can calculate the frequency distribution of the top 50 words. Initially, we started by using no filter. On seeing the presence of non-alphabetical characters, I used regular expression to remove them. Still the list contained many stop words which could be ignored. On removing the stop words list from NLTK I obtained a much cleaner set of words for the frequency list. Also, I created my own list of stop words which included two words to obtain better results.

The new list of top 50 frequent words on applying the new stop words list is

```
In [117]: stoppedflightwords = [w for w in flightalphawords if not w in wordsFiltered]
          # Print the updated list of frequencies without stop words
          flightdist = FreqDist(stoppedflightwords)
          flightitems = flightdist.most_common(50)
          for item in flightitems:
              print(item)
```

```
('machine', 170)
('project', 87)
('would', 74)
('one', 72)
('wind', 70)
('work', 60)
('gutenberg-tm', 56)
('feet', 52)
('flight', 50)
('motor', 47)
('per', 44)
('miles', 43)
('could', 42)
('first', 39)
('flying', 39)
('surfaces', 39)
('time', 37)
('speed', 36)
('experiments', 35)
('works', 34)
('hour', 33)
('air', 32)
('found', 31)
('angle', 31)
('operator', 31)
('ground', 31)
('gutenberg', 30)
('power', 30)
```

Now we can observe that we obtained a much better frequency list with meaningful words when compared to the initial results.

- Bigrams – A bigram is a pair of consecutive written units. I used nltk.collocations package here as we get more functionalities with regard to Bigrams. The collocations are basically expressions of multiple words which commonly co-occur. The collocations have a finder which allows us to call other functions and give scores to the bigrams.
  Initially I started by not applying any filters to obtain the bigrams along with its scores. The list consisted of bigram which had non-alphabetical characters which I removed using regular expressions. Since this list consisted of words where there were no stop words I did not have to add any other filters

```
In [97]: finder.apply_word_filter(alpha_filter)
         scored = finder.score_ngrams(bigram_measure.raw_freq)
         for bscore in scored[:50]:
             print(bscore)
```

```
(('project', 'gutenberg-tm'), 0.0028821410190427174)
(('project', 'gutenberg'), 0.001544004117344313)
(('gutenberg-tm', 'electronic'), 0.0009264024704065877)
(('electronic', 'works'), 0.0008234688625836336)
(('per', 'hour'), 0.0007720020586721565)
(('miles', 'per'), 0.0007205352547606794)
(('archive', 'foundation'), 0.0006690684508492022)
(('gutenberg', 'literary'), 0.0006690684508492022)
(('literary', 'archive'), 0.0006690684508492022)
(('set', 'forth'), 0.0006176016469377251)
(('united', 'states'), 0.0006176016469377251)
(('electronic', 'work'), 0.000566134843026248)
(('flying', 'machine'), 0.000566134843026248)
(('kill', 'devil'), 0.000514668039114771)
(('kitty', 'hawk'), 0.000514668039114771)
(('mr.', 'chanute'), 0.000514668039114771)
(('square', 'feet'), 0.0004632012352032938 5)
(('gutenberg-tm', 'license'), 0.0004117344312918168)
(('head', 'resistance'), 0.0004117344312918168)
(('per', 'cent'), 0.0004117344312918168)
(('per', 'second'), 0.0004117344312918168)
(('public', 'domain'), 0.0004117344312918168)
(('devil', 'hill'), 0.0003602676273803397)
(('full', 'project'), 0.0003602676273803397)
(('three', 'degrees'), 0.0003602676273803397)
(('curved', 'surfaces'), 0.00030880082346886257)
(('early', 'history'), 0.00030880082346886257)
(('first', 'flight'), 0.00030880082346886257)
(('flying', 'machines'), 0.00030880082346886257)
(('horizontal', 'position'), 0.00030880082346886257)
(('hundred', 'feet'), 0.00030880082346886257)
(("'s", 'body'), 0.0002573340195573855)
(('feet', 'per'), 0.0002573340195573855)
(('gutenberg-tm', 'work'), 0.0002573340195573855)
(('gutenberg-tm', 'works'), 0.0002573340195573855)
```

.

- Mutual Information – PMI scores have been used for finding collocations and association between words

With the obtained finder and scorer, we compute the Pointwise mutual information score without applying any filters. We set a minimum frequency for this text as 5 which will enable us to get bigrams which make more sense.

The list generated with PMI scores have non-alphabetical characters. We can remove them by applying the function we defined using regular expressions.
This list generated consists of no stopwords and hence we don't need to apply the stop words filter in this case. There are a few bigrams which have word lengths less than 2 and do not make sense. We can apply a filter to remove words whose length is small.
Here we have a condition which removes words in w1 and w2 whose length is lesser than or equal to 2

```
In [129]: finder3.apply_ngram_filter(lambda w1,w2: len(w1)<=2 & len(w2)<=2)
          scored = finder3.score_ngrams(bigram_measure.pmi)
          for bscore in scored[:50]:
              print(bscore)
```

```
(('professor', 'langley'), 11.17560895258131)
(('public', 'domain'), 10.924070185585347)
(('kill', 'devil'), 10.924070185585345)
(('kitty', 'hawk'), 10.78656666183541)
(('early', 'history'), 10.66103577975155)
(('literary', 'archive'), 10.545558562331614)
(('united', 'states'), 10.245998280472707)
(('head', 'resistance'), 9.96059606161046)
(('set', 'forth'), 9.924070185585343)
(('devil', 'hill'), 9.643962266392611)
(('archive', 'foundation'), 9.545558562331614)
(('gutenberg', 'literary'), 9.339107684864189)
(('mr.', 'chanute'), 8.874439417860744)
(('per', 'cent'), 8.786566661835412)
(('horizontal', 'position'), 8.754145184143031)
(('three', 'degrees'), 8.610409706681585)
(('electronic', 'works'), 8.4036479370589)
(('curved', 'surfaces'), 8.375633560889302)
(('information', 'about'), 8.201604161114252)
(('hundred', 'feet'), 8.13052106305277)
(('square', 'feet'), 7.908128641716324)
(('gutenberg-tm', 'electronic'), 7.8536808576939485)
(('project', 'gutenberg-tm'), 7.803054784623981)
(('project', 'gutenberg'), 7.803054784623978)
(('would', 'develop'), 7.773510509009965)
(('per', 'second'), 7.699103820585071)
(('per', 'hour'), 7.649063138085477)
(('flying', 'machines'), 7.638667966723094)
(('gutenberg-tm', 'license'), 7.531752762806585)
(('less', 'than'), 7.495570426499938)
(('operator', "'s"), 7.443805063530883)
(('little', 'over'), 7.228076372475446)
(('did', 'not'), 7.211574449869181)
```

For Indian book –

- Tokenization – I performed tokenization because this book consisted of many unnecessary characters and punctuations. Also, with tokenization we can chop of each word into smaller units.

```
In [37]: #perform tokenization

         indiaTokens = nltk.word_tokenize(text)
```

- Lowercase – I converted the tokenized text into lower case to make all the words accept to a universal case format. This would ensure that the system is not in a dilemma when it considers “The” and “the”. I also printed the length of the tokenized words after converting them to lower case.

```
In [38]: indiawords = [w.lower() for w in indiaTokens]
```

```
In [39]: #print length of tokenized text

         print(len(indiawords))
```

```
163739
```

- Frequency of top 50 words – By using from nltk import FreqDist we can calculate the frequency distribution of the top 50 words. Initally we started by using no filter. On seeing the presence of non-alphabetical characters, I used regular expression to remove them. Still the list contained many stop words which could be ignored. On removing the stop words list from NLTK I obtained a much cleaner set of words for the frequency list. Also, I created my own list of stop words which included two words to obtain better results.

From the above frequency table, we infer that our results are getting better and better by applying the filters. Now lets try to get a better solution by adding to more common words into a new stopword list called wordsFiltered. This list is initially empty and we will add the current stopwords list from NLTK to it and add two more words which are “made”, “much”, “may”, “many”, “even” to our list and generate the top 50 frequent words.

The new list of top 50 frequent words on applying the new stop words list is

```
In [51]: stoppedindiawords = [w for w in indiaalphawords if not w in wordsFiltered]
         # Print the updated list of frequencies without stop words
         indiadist = FreqDist(stoppedindiawords)
         indiaitems = indiadist.most_common(50)
         for item in indiaitems:
             print(item)
```

```
('india', 734)
('one', 481)
('upon', 439)
('people', 292)
('great', 281)
('government', 265)
('every', 249)
('years', 241)
('native', 226)
('several', 224)
('men', 210)
('city', 185)
('feet', 180)
('would', 168)
('time', 166)
('women', 164)
('two', 161)
('hindu', 160)
('bombay', 150)
('work', 150)
('british', 149)
('country', 137)
('like', 137)
('without', 137)
('long', 134)
("'s", 131)
('man', 128)
```

Now we can observe that we obtained a much better frequency list with meaningful words when compared to the initial results.

- Bigrams – A bigram is a pair of consecutive written units. I used nltk.collocations package here as we get more functionalities with regard to Bigrams. The collocations are basically expressions of multiple words which commonly co-occur. The collocations have a finder which allows us to call other functions and give scores to the bigrams.
  Initially I started by not applying any filters to obtain the bigrams along with its scores. The list consisted of bigrams which had non-alphabetical characters which I removed using regular expressions. Since this list consisted of stop words I used my self-created stop word list to remove them.

```
In [56]: finder.apply_word_filter(lambda w: w in wordsFiltered)
         scored = finder.score_ngrams(bigram_measure.raw_freq)
         for bscore in scored[:50]:
             print(bscore)
```

```
(('per', 'cent'), 0.0004824751586366107)
(('lord', 'curzon'), 0.0003420077073879772)
(('project', 'gutenberg-tm'), 0.0003420077073879772)
(('united', 'states'), 0.00032979314640983515)
(('ten', 'years'), 0.00025039850005191186)
(('years', 'ago'), 0.00021375481711748576)
(('feet', 'high'), 0.00020764753662841473)
(('native', 'princes'), 0.00018321841467213064)
(('project', 'gutenberg'), 0.00018321841467213064)
(('lady', 'curzon'), 0.00015878929271584657)
(('native', 'states'), 0.00015268201222677554)
(('shah', 'jehan'), 0.0001404674512486335)
(('british', 'government'), 0.00013436017075956248)
(('white', 'marble'), 0.00013436017075956248)
(('northern', 'india'), 0.00012825289027049145)
(('precious', 'stones'), 0.00012825289027049145)
(('taj', 'mahal'), 0.00012825289027049145)
(('east', 'india'), 0.00012214560978142044)
(('feet', 'wide'), 0.00011603832929234941)
(('india', 'company'), 0.00011603832929234941)
(('gutenberg-tm', 'electronic'), 0.00010993104880327838)
(('five', 'years'), 0.00010382376831420737)
(('electronic', 'works'), 9.771648782513635e-05)
(('great', 'deal'), 9.771648782513635e-05)
(('feet', 'long'), 9.160920733606532e-05)
(('years', 'old'), 9.160920733606532e-05)
(('dependent', 'upon'), 8.55019268469943e-05)
(('last', 'ten'), 8.55019268469943e-05)
(('archive', 'foundation'), 7.939464635792329e-05)
(('government', 'house'), 7.939464635792329e-05)
(('gutenberg', 'literary'), 7.939464635792329e-05)
(('hundred', 'years'), 7.939464635792329e-05)
```

- Mutual Information – PMI scores have been used for finding collocations and association between words

With the obtained finder and scorer, we compute the Pointwise mutual information score without applying any filters. We set a minimum frequency for this text as 5 which will enable us to get bigrams which make more sense.
The list generated with PMI scores have non-alphabetical characters. We can remove them by applying the function we defined using regular expressions. Since the list does not contain stop words I did not have use the stop filter to remove them.

In [59]:
```
finder3.apply_word_filter(alpha_filter)
scored = finder3.score_ngrams(bigram_measure.pmi)
for bscore in scored[:50]:
    print(bscore)
```

```
(('melting', 'snows'), 14.999110369078931)
(('kutab', 'minar'), 14.736075963245135)
(('warren', 'hastings'), 14.321038463966289)
(('prime', 'minister'), 14.151113462523977)
(('bullock', 'carts'), 13.835611636796049)
(('nana', 'sahib'), 13.584072869800085)
(('edward', 'vii'), 13.513683541908687)
(('funeral', 'pyres'), 13.513683541908687)
(('austin', 'de'), 13.473041557411342)
(('literary', 'archive'), 13.414147868357773)
(('nautch', 'dancers'), 13.291291120572238)
(('funeral', 'pyre'), 13.028256714738445)
(('aga', 'khan'), 12.92872104118753)
(('fellow', 'countrymen'), 12.88046587258031)
(('colonel', 'younghusband'), 12.719002449886192)
(('archive', 'foundation'), 12.513683541908687)
(('taj', 'mahal'), 12.489354212546948)
(('nur', 'jehan'), 12.414147868357773)
(('gutenberg', 'literary'), 12.207696990890346)
(('queen', 'victoria'), 12.179127838693574)
(('red', 'sandstone'), 12.10804474063209)
(('khyber', 'pass'), 12.073110950522707)
(('precious', 'stones'), 12.058719858217085)
(('burning', 'ghat'), 11.969363025684874)
(('mechanical', 'industries'), 11.942526840712562)
(('industrial', 'arts'), 11.88046587258031)
(('brass', 'bowl'), 11.77260183927025)
(('saracenic', 'architecture'), 11.741094038011758)
(('arabian', 'sea'), 11.736075963245135)
(('grand', 'lama'), 11.666402435438322)
(('savings', 'banks'), 11.665686635353737)
(('nadir', 'shah'), 11.5985724394952)
(('lieutenant', 'governor'), 11.56615096180282)
(('previous', 'chapter'), 11.521063072274284)
(('middle', 'ages'), 11.513683541908687)
```

2. b) Yes, there were problems which I encountered in the bigram lists.
   For the flight book –
   Initially I started off making the bigram lists without using any filters. Without using any filters, I obtained a bigram list which consisted of non-alphabetical characters as shown below.

```
In [86]: from nltk.collocations import *

In [87]: bigram_measure = nltk.collocations.BigramAssocMeasures()

In [89]: finder = BigramCollocationFinder.from_words(flightwords)
         scored = finder.score_ngrams(bigram_measure.raw_freq)

In [92]: for bscore in scored[:50]:
             print(bscore)
```

```
(('of', 'the'), 0.01291816778178075)
((',', 'and'), 0.007050952135872362)
(('the', 'machine'), 0.005455481214616573)
(('to', 'the'), 0.0051466803911477095)
(('.', 'the'), 0.005095213587236232)
(('in', 'the'), 0.004529078744209984)
(('--', '--'), 0.0032938754503345343)
((',', 'the'), 0.0029336078229541943)
((',', 'we'), 0.0028821410190427174)
(('project', 'gutenberg-tm'), 0.0028821410190427174)
(('.', 'we'), 0.002624806999485332)
(('with', 'the'), 0.002264539372104992)
((',', 'but'), 0.0021616057642820383)
(('on', 'the'), 0.0021616057642820383)
(('that', 'the'), 0.0021616057642820383)
(('and', 'the'), 0.002058672156459084)
(('the', 'wind'), 0.00195573854863613)
(('we', 'were'), 0.00195573854863613)
(('of', 'this'), 0.0018013381369016985)
(('from', 'the'), 0.0016984045290787443)
(('in', 'a'), 0.0016469377251672672)
(('we', 'had'), 0.0016469377251672672)
(('of', 'a'), 0.00159547092125579)
(('the', 'project'), 0.00159547092125579)
(('project', 'gutenberg'), 0.001544004117344313)
(('by', 'the'), 0.0014925373134328358)
(('the', 'motor'), 0.0014925373134328358)
(('the', 'operator'), 0.0014410705095213587)
```

To remove the non-alphabetical characters, I used the function which we defined earlier using regular expressions. The list without non-alphabetical characters are shown below. This list provides more sense

In [97]:
```
finder.apply_word_filter(alpha_filter)
scored = finder.score_ngrams(bigram_measure.raw_freq)
for bscore in scored[:50]:
    print(bscore)
```

```
(('project', 'gutenberg-tm'), 0.0028821410190427174)
(('project', 'gutenberg'), 0.001544004117344313)
(('gutenberg-tm', 'electronic'), 0.0009264024704065877)
(('electronic', 'works'), 0.0008234688625836336)
(('per', 'hour'), 0.0007720020586721565)
(('miles', 'per'), 0.0007205352547606794)
(('archive', 'foundation'), 0.0006690684508492022)
(('gutenberg', 'literary'), 0.0006690684508492022)
(('literary', 'archive'), 0.0006690684508492022)
(('set', 'forth'), 0.0006176016469377251)
(('united', 'states'), 0.0006176016469377251)
(('electronic', 'work'), 0.000566134843026248)
(('flying', 'machine'), 0.000566134843026248)
(('kill', 'devil'), 0.000514668039114771)
(('kitty', 'hawk'), 0.000514668039114771)
(('mr.', 'chanute'), 0.000514668039114771)
(('square', 'feet'), 0.0004632012352032938 5)
(('gutenberg-tm', 'license'), 0.0004117344312918168)
(('head', 'resistance'), 0.0004117344312918168)
(('per', 'cent'), 0.0004117344312918168)
(('per', 'second'), 0.0004117344312918168)
(('public', 'domain'), 0.0004117344312918168)
(('devil', 'hill'), 0.0003602676273803397)
(('full', 'project'), 0.0003602676273803397)
(('three', 'degrees'), 0.0003602676273803397)
(('curved', 'surfaces'), 0.0003088008234688625 7)
(('early', 'history'), 0.0003088008234688625 7)
(('first', 'flight'), 0.0003088008234688625 7)
(('flying', 'machines'), 0.0003088008234688625 7)
(('horizontal', 'position'), 0.0003088008234688625 7)
(('hundred', 'feet'), 0.0003088008234688625 7)
(("'s", 'body'), 0.0002573340195573855)
(('feet', 'per'), 0.0002573340195573855)
(('gutenberg-tm', 'work'), 0.0002573340195573855)
(('gutenberg-tm', 'works'), 0.0002573340195573855)
```

This is the best bigram list I obtained using raw_freq. I did not need to apply stop words to this list as this list did not consist of any stop words

In Mutual Information, I sent the minimum freq to 5 and calculated the bigrams without any filters

In [102]:
```
finder3.apply_freq_filter(5)
scored = finder3.score_ngrams(bigram_measure.pmi)
for bscore in scored[:50]:
    print(bscore)
```

```
(('professor', 'langley'), 11.17560895258131)
(('public', 'domain'), 10.924070185585347)
(('kill', 'devil'), 10.924070185585345)
(('kitty', 'hawk'), 10.78656666183541)
(('[', 'illustration'), 10.66103577975155)
(('early', 'history'), 10.66103577975155)
(('illustration', ']'), 10.66103577975155)
(('literary', 'archive'), 10.545558562331614)
(('united', 'states'), 10.245998280472707)
(('head', 'resistance'), 9.96059606161046)
(('set', 'forth'), 9.924070185585343)
(('devil', 'hill'), 9.643962266392611)
(('archive', 'foundation'), 9.545558562331614)
(('gutenberg', 'literary'), 9.339107684864189)
(("'s", 'body'), 9.228076372475446)
(('http', ':'), 9.158535439222367)
(('*', '*'), 9.017179589976825)
(('mr.', 'chanute'), 8.874439417860744)
(('per', 'cent'), 8.786566661835412)
(('horizontal', 'position'), 8.754145184143031)
(('three', 'degrees'), 8.610409706681585)
(('electronic', 'works'), 8.4036479370589)
(('curved', 'surfaces'), 8.375633560889302)
(('information', 'about'), 8.201604161114252)
(('(', '2'), 8.145861609187259)
(('2', ')'), 8.145861609187259)
(('hundred', 'feet'), 8.13052106305277)
(('(', '3'), 7.979211739777808)
(('3', ')'), 7.979211739777808)
(('(', '1'), 7.953216531244863)
(('1', ')'), 7.953216531244863)
(('square', 'feet'), 7.908128641716324)
(('gutenberg-tm', 'electronic'), 7.8536808576939485)
```

This list consisted of a lot of non-alphabetical characters, hence I used the alpha_filter function to remove them and then obtained

In [124]:
```
finder3.apply_word_filter(alpha_filter)
scored = finder3.score_ngrams(bigram_measure.pmi)
for bscore in scored[:50]:
    print(bscore)
```

```
(('professor', 'langley'), 11.17560895258131)
(('public', 'domain'), 10.924070185585347)
(('kill', 'devil'), 10.924070185585345)
(('kitty', 'hawk'), 10.78656666183541)
(('early', 'history'), 10.66103577975155)
(('literary', 'archive'), 10.545558562331614)
(('united', 'states'), 10.245998280472707)
(('head', 'resistance'), 9.96059606161046)
(('set', 'forth'), 9.924070185585343)
(('devil', 'hill'), 9.643962266392611)
(('archive', 'foundation'), 9.545558562331614)
(('gutenberg', 'literary'), 9.339107684864189)
(("'s", 'body'), 9.228076372475446)
(('mr.', 'chanute'), 8.874439417860744)
(('per', 'cent'), 8.786566661835412)
(('horizontal', 'position'), 8.754145184143031)
(('three', 'degrees'), 8.610409706681585)
(('electronic', 'works'), 8.4036479370589)
(('curved', 'surfaces'), 8.375633560889302)
(('information', 'about'), 8.201604161114252)
(('hundred', 'feet'), 8.13052106305277)
(('square', 'feet'), 7.908128641716324)
(('gutenberg-tm', 'electronic'), 7.8536808576939485)
(('project', 'gutenberg-tm'), 7.803054784623981)
(('project', 'gutenberg'), 7.803054784623978)
(('would', 'develop'), 7.773510509009965)
(('per', 'second'), 7.699103820585071)
(('per', 'hour'), 7.649063138085477)
(('flying', 'machines'), 7.638667966723094)
(('gutenberg-tm', 'license'), 7.531752762806585)
(('less', 'than'), 7.495570426499938)
(('operator', "'s"), 7.443805063530883)
(('little', 'over'), 7.228076372475446)
```

Now, since there are words which have a length less than 2 I applied a filter to remove words whose length is less than or equal to 2. The list obtained is

In [129]:
```
finder3.apply_ngram_filter(lambda w1,w2: len(w1)<=2 & len(w2)<=2)
scored = finder3.score_ngrams(bigram_measure.pmi)
for bscore in scored[:50]:
    print(bscore)
```

```
(('professor', 'langley'), 11.17560895258131)
(('public', 'domain'), 10.924070185585347)
(('kill', 'devil'), 10.924070185585345)
(('kitty', 'hawk'), 10.78656666183541)
(('early', 'history'), 10.66103577975155)
(('literary', 'archive'), 10.545558562331614)
(('united', 'states'), 10.245998280472707)
(('head', 'resistance'), 9.96059606161046)
(('set', 'forth'), 9.924070185585343)
(('devil', 'hill'), 9.643962266392611)
(('archive', 'foundation'), 9.545558562331614)
(('gutenberg', 'literary'), 9.339107684864189)
(('mr.', 'chanute'), 8.874439417860744)
(('per', 'cent'), 8.786566661835412)
(('horizontal', 'position'), 8.754145184143031)
(('three', 'degrees'), 8.610409706681585)
(('electronic', 'works'), 8.4036479370589)
(('curved', 'surfaces'), 8.375633560889302)
(('information', 'about'), 8.201604161114252)
(('hundred', 'feet'), 8.13052106305277)
(('square', 'feet'), 7.908128641716324)
(('gutenberg-tm', 'electronic'), 7.8536808576939485)
(('project', 'gutenberg-tm'), 7.803054784623981)
(('project', 'gutenberg'), 7.803054784623978)
(('would', 'develop'), 7.773510509009965)
(('per', 'second'), 7.699103820585071)
(('per', 'hour'), 7.649063138085477)
(('flying', 'machines'), 7.638667966723094)
(('gutenberg-tm', 'license'), 7.531752762806585)
(('less', 'than'), 7.495570426499938)
(('operator', "'s"), 7.443805063530883)
(('little', 'over'), 7.228076372475446)
(('did', 'not'), 7.211574449869181)
```

For the Indian book –

Initially I started off making the bigram lists without using any filters. Without using any filters, I obtained a bigram list which consisted of non-alphabetical characters as shown below.

```
In [16]: from nltk.collocations import *

In [17]: bigram_measure = nltk.collocations.BigramAssocMeasures()

In [53]: finder = BigramCollocationFinder.from_words(indiawords)
         scored = finder.score_ngrams(bigram_measure.raw_freq)

In [54]: for bscore in scored[:50]:
             print(bscore)
```

```
(('of', 'the'), 0.013472660758890673)
((',', 'and'), 0.012428315795259529)
(('.', 'the'), 0.006650828452598343)
(('in', 'the'), 0.005649234452390695)
(('and', 'the'), 0.004097985208166655)
((',', 'the'), 0.004030805122786875)
(('to', 'the'), 0.0031880004152950734)
((',', 'but'), 0.0027788126225273147)
(('it', 'is'), 0.0027604907810601016)
((',', 'which'), 0.0022352646589999937)
(('.', 'it'), 0.0021619772931311416)
((',', 'who'), 0.001716145817428957)
(('by', 'the'), 0.001685609414983602)
(('for', 'the'), 0.001685609414983602)
(('from', 'the'), 0.0016672875735163888)
(('in', 'india'), 0.0016123220491147496)
(('.', 'they'), 0.0016001074881366077)
(('of', 'india'), 0.0016001074881366077)
(('the', 'most'), 0.0015451419637349684)
(('they', 'are'), 0.0015451419637349684)
(('one', 'of'), 0.0014962837198224002)
(('is', 'a'), 0.0014413181954207612)
(('.', 'there'), 0.0013008507441721277)
(('of', 'his'), 0.0012825289027049145)
(('.', 'he'), 0.0012580997807486304)
(('of', 'a'), 0.0012336706587923463)
(('and', 'other'), 0.0012214560978142043)
(('on', 'the'), 0.0012214560978142043)
(('with', 'the'), 0.0012153488173251333)
```

From this list, there exists many non-alphabetical characters. I added a filter to remove these characters using a function defined above call alpha_filter which uses regular expressions

In [55]:
```
finder.apply_word_filter(alpha_filter)
scored = finder.score_ngrams(bigram_measure.raw_freq)
for bscore in scored[:50]:
    print(bscore)
```

```
(('of', 'the'), 0.013472660758890673)
(('in', 'the'), 0.005649234452390695)
(('and', 'the'), 0.004097985208166655)
(('to', 'the'), 0.0031880004152950734)
(('it', 'is'), 0.0027604907810601016)
(('by', 'the'), 0.001685609414983602)
(('for', 'the'), 0.001685609414983602)
(('from', 'the'), 0.0016672875735163888)
(('in', 'india'), 0.0016123220491147496)
(('of', 'india'), 0.0016001074881366077)
(('the', 'most'), 0.0015451419637349684)
(('they', 'are'), 0.0015451419637349684)
(('one', 'of'), 0.0014962837198224002)
(('is', 'a'), 0.0014413181954207612)
(('of', 'his'), 0.0012825289027049145)
(('of', 'a'), 0.0012336706587923463)
(('and', 'other'), 0.0012214560978142043)
(('on', 'the'), 0.0012214560978142043)
(('with', 'the'), 0.0012153488173251333)
(('at', 'the'), 0.001142061451456281)
(('is', 'the'), 0.001105417768521855)
(('in', 'a'), 0.0010870959270546418)
(('have', 'been'), 0.0010687740855874288)
(('to', 'be'), 0.0010626668050983576)
(('has', 'been'), 0.0010565595246092867)
(('upon', 'the'), 0.0010321304026530025)
(('that', 'the'), 0.0010138085611857896)
(('the', 'government'), 0.0009954867197185765)
(('the', 'same'), 0.0009649503172732214)
(('there', 'is'), 0.0009405211953169374)
(('there', 'are'), 0.0008977702318934402)
(('of', 'their'), 0.000848911987980872)
(('of', 'them'), 0.0008122683050464459)
(('it', 'was'), 0.0007939464635792329)
```

Since the above list consists of many stopwords I made another filter to remove the stop words and to generate a list which makes more sense.

In [56]:
```
finder.apply_word_filter(lambda w: w in wordsFiltered)
scored = finder.score_ngrams(bigram_measure.raw_freq)
for bscore in scored[:50]:
    print(bscore)
```

```
(('per', 'cent'), 0.0004824751586366107)
(('lord', 'curzon'), 0.0003420077073879772)
(('project', 'gutenberg-tm'), 0.0003420077073879772)
(('united', 'states'), 0.00032979314640983515)
(('ten', 'years'), 0.00025039850005191186)
(('years', 'ago'), 0.00021375481711748576)
(('feet', 'high'), 0.00020764753662841473)
(('native', 'princes'), 0.00018321841467213064)
(('project', 'gutenberg'), 0.00018321841467213064)
(('lady', 'curzon'), 0.00015878929271584657)
(('native', 'states'), 0.00015268201222677554)
(('shah', 'jehan'), 0.0001404674512486335)
(('british', 'government'), 0.00013436017075956248)
(('white', 'marble'), 0.00013436017075956248)
(('northern', 'india'), 0.00012825289027049145)
(('precious', 'stones'), 0.00012825289027049145)
(('taj', 'mahal'), 0.00012825289027049145)
(('east', 'india'), 0.00012214560978142044)
(('feet', 'wide'), 0.00011603832929234941)
(('india', 'company'), 0.00011603832929234941)
(('gutenberg-tm', 'electronic'), 0.00010993104880327838)
(('five', 'years'), 0.00010382376831420737)
(('electronic', 'works'), 9.771648782513635e-05)
(('great', 'deal'), 9.771648782513635e-05)
(('feet', 'long'), 9.160920733606532e-05)
(('years', 'old'), 9.160920733606532e-05)
(('dependent', 'upon'), 8.55019268469943e-05)
(('last', 'ten'), 8.55019268469943e-05)
(('archive', 'foundation'), 7.939464635792329e-05)
(('government', 'house'), 7.939464635792329e-05)
(('gutenberg', 'literary'), 7.939464635792329e-05)
(('hundred', 'years'), 7.939464635792329e-05)
```

Now using Pointwise Mutual information, I obtained the following results without using any filters and setting the minimum freq to 5

In [58]:
```
finder3.apply_freq_filter(5)
scored = finder3.score_ngrams(bigram_measure.pmi)
for bscore in scored[:50]:
    print(bscore)
```

```
(('melting', 'snows'), 14.999110369078931)
(('kutab', 'minar'), 14.736075963245135)
(('warren', 'hastings'), 14.321038463966289)
(('prime', 'minister'), 14.151113462523977)
(('bullock', 'carts'), 13.835611636796049)
(('nana', 'sahib'), 13.584072869800085)
(('edward', 'vii'), 13.513683541908687)
(('funeral', 'pyres'), 13.513683541908687)
(('austin', 'de'), 13.473041557411342)
(('literary', 'archive'), 13.414147868357773)
(('nautch', 'dancers'), 13.291291120572238)
(('funeral', 'pyre'), 13.028256714738445)
(('aga', 'khan'), 12.92872104118753)
(('fellow', 'countrymen'), 12.88046587258031)
(('colonel', 'younghusband'), 12.719002449886192)
(('archive', 'foundation'), 12.513683541908687)
(('taj', 'mahal'), 12.489354212546948)
(('nur', 'jehan'), 12.414147868357773)
(('gutenberg', 'literary'), 12.207696990890346)
(('queen', 'victoria'), 12.179127838693574)
(('red', 'sandstone'), 12.10804474063209)
(('khyber', 'pass'), 12.073110950522707)
(('[', 'illustration'), 12.0609156535524)
(('precious', 'stones'), 12.058719858217085)
(('burning', 'ghat'), 11.969363025684874)
(('mechanical', 'industries'), 11.942526840712562)
(('industrial', 'arts'), 11.88046587258031)
(('brass', 'bowl'), 11.77260183927025)
(('saracenic', 'architecture'), 11.741094038011758)
(('arabian', 'sea'), 11.736075963245135)
(('grand', 'lama'), 11.666402435438322)
(('savings', 'banks'), 11.665686635353737)
```

This list made good sense, but consisted of non-alphabetical characters which I removed using the alpha_filter function

In [59]:
```
finder3.apply_word_filter(alpha_filter)
scored = finder3.score_ngrams(bigram_measure.pmi)
for bscore in scored[:50]:
    print(bscore)
```

```
(('melting', 'snows'), 14.999110369078931)
(('kutab', 'minar'), 14.736075963245135)
(('warren', 'hastings'), 14.321038463966289)
(('prime', 'minister'), 14.151113462523977)
(('bullock', 'carts'), 13.835611636796049)
(('nana', 'sahib'), 13.584072869800085)
(('edward', 'vii'), 13.513683541908687)
(('funeral', 'pyres'), 13.513683541908687)
(('austin', 'de'), 13.473041557411342)
(('literary', 'archive'), 13.414147868357773)
(('nautch', 'dancers'), 13.291291120572238)
(('funeral', 'pyre'), 13.028256714738445)
(('aga', 'khan'), 12.92872104118753)
(('fellow', 'countrymen'), 12.88046587258031)
(('colonel', 'younghusband'), 12.719002449886192)
(('archive', 'foundation'), 12.513683541908687)
(('taj', 'mahal'), 12.489354212546948)
(('nur', 'jehan'), 12.414147868357773)
(('gutenberg', 'literary'), 12.207696990890346)
(('queen', 'victoria'), 12.179127838693574)
(('red', 'sandstone'), 12.10804474063209)
(('khyber', 'pass'), 12.073110950522707)
(('precious', 'stones'), 12.058719858217085)
(('burning', 'ghat'), 11.969363025684874)
(('mechanical', 'industries'), 11.942526840712562)
(('industrial', 'arts'), 11.88046587258031)
(('brass', 'bowl'), 11.77260183927025)
(('saracenic', 'architecture'), 11.741094038011758)
(('arabian', 'sea'), 11.736075963245135)
(('grand', 'lama'), 11.666402435438322)
(('savings', 'banks'), 11.665686635353737)
(('nadir', 'shah'), 11.5985724394952)
(('lieutenant', 'governor'), 11.56615096180282)
(('previous', 'chapter'), 11.521063072274284)
(('middle', 'ages'), 11.513683541908687)
```

This list is the best solution I obtained for PMI.

The lists obtained from PMI and raw frequency are different as

For Flight book –
The scores obtained for raw frequency ranges from 0 and 1.

```
(('project', 'gutenberg-tm'), 0.0028821410190427174)
(('project', 'gutenberg'), 0.0015440041173443 13)
(('gutenberg-tm', 'electronic'), 0.0009264024704065877)
(('electronic', 'works'), 0.0008234688625836336)
(('per', 'hour'), 0.0007720020586721565)
(('miles', 'per'), 0.0007205352547606794)
(('archive', 'foundation'), 0.0006690684508492022)
(('gutenberg', 'literary'), 0.0006690684508492022)
(('literary', 'archive'), 0.0006690684508492022)
(('set', 'forth'), 0.0006176016469377251)
(('united', 'states'), 0.0006176016469377251)
(('electronic', 'work'), 0.000566134843026248)
(('flying', 'machine'), 0.000566134843026248)
(('kill', 'devil'), 0.0005146680391147 71)
(('kitty', 'hawk'), 0.0005146680391147 71)
(('mr.', 'chanute'), 0.0005146680391147 71)
(('square', 'feet'), 0.0004632012352032 9385)
(('gutenberg-tm', 'license'), 0.0004117344312918168)
(('head', 'resistance'), 0.0004117344312918168)
(('per', 'cent'), 0.0004117344312918168)
(('per', 'second'), 0.0004117344312918168)
(('public', 'domain'), 0.0004117344312918168)
```

The scores obtained for PMI scores range from 5-12

```
(('professor', 'langley'), 11.17560895258131)
(('public', 'domain'), 10.924070185585347)
(('kill', 'devil'), 10.924070185585345)
(('kitty', 'hawk'), 10.78656666183541)
(('early', 'history'), 10.66103577975155)
(('literary', 'archive'), 10.545558562331614)
(('united', 'states'), 10.2459982804727 07)
(('head', 'resistance'), 9.96059606161046)
(('set', 'forth'), 9.924070185585343)
(('devil', 'hill'), 9.643962266392611)
(('archive', 'foundation'), 9.545558562331614)
(('gutenberg', 'literary'), 9.339107684864189)
(('mr.', 'chanute'), 8.874439417860744)
(('per', 'cent'), 8.786566661835412)
(('horizontal', 'position'), 8.754145184143031)
(('three', 'degrees'), 8.610409706681585)
(('electronic', 'works'), 8.4036479370589)
(('curved', 'surfaces'), 8.375633560889302)
(('information', 'about'), 8.201604161114252)
(('hundred', 'feet'), 8.13052106305277)
```

Both the Bigrams gave sensible information and included words which made sense, but PMI gave words which made more technical meaning when compared to raw_freq.

For Indian book

The scores obtained for raw frequency ranges from 0 and 10.

```
(('per', 'cent'), 0.0004824751586366107)
(('lord', 'curzon'), 0.0003420077073879772)
(('project', 'gutenberg-tm'), 0.0003420077073879772)
(('united', 'states'), 0.00032979314640983515)
(('ten', 'years'), 0.00025039850005191186)
(('years', 'ago'), 0.00021375481711748576)
(('feet', 'high'), 0.00020764753662841473)
(('native', 'princes'), 0.00018321841467213064)
(('project', 'gutenberg'), 0.00018321841467213064)
(('lady', 'curzon'), 0.00015878929271584657)
(('native', 'states'), 0.00015268201222677554)
(('shah', 'jehan'), 0.0001404674512486335)
(('british', 'government'), 0.00013436017075956248)
(('white', 'marble'), 0.00013436017075956248)
(('northern', 'india'), 0.00012825289027049145)
(('precious', 'stones'), 0.00012825289027049145)
(('taj', 'mahal'), 0.00012825289027049145)
(('east', 'india'), 0.00012214560978142044)
(('feet', 'wide'), 0.00011603832929234941)
(('india', 'company'), 0.00011603832929234941)
(('gutenberg-tm', 'electronic'), 0.00010993104880327838)
(('five', 'years'), 0.00010382376831420737)
(('electronic', 'works'), 9.771648782513635e-05)
(('great', 'deal'), 9.771648782513635e-05)
(('feet', 'long'), 9.160920733606532e-05)
(('years', 'old'), 9.160920733606532e-05)
(('dependent', 'upon'), 8.55019268469943e-05)
(('last', 'ten'), 8.55019268469943e-05)
(('archive', 'foundation'), 7.939464635792329e-05)
(('government', 'house'), 7.939464635792329e-05)
(('gutenberg', 'literary'), 7.939464635792329e-05)
(('hundred', 'years'), 7.939464635792329e-05)
(('literary', 'archive'), 7.939464635792329e-05)
```

The scores obtained for PMI scores range from 10-15

```
(('melting', 'snows'), 14.999110369078931)
(('kutab', 'minar'), 14.736075963245135)
(('warren', 'hastings'), 14.321038463966289)
(('prime', 'minister'), 14.151113462523977)
(('bullock', 'carts'), 13.835611636796049)
(('nana', 'sahib'), 13.5840728698000 85)
(('edward', 'vii'), 13.513683541908687)
(('funeral', 'pyres'), 13.513683541908687)
(('austin', 'de'), 13.473041557411342)
(('literary', 'archive'), 13.414147868357773)
(('nautch', 'dancers'), 13.291291120572238)
(('funeral', 'pyre'), 13.028256714738445)
(('aga', 'khan'), 12.92872104118753)
(('fellow', 'countrymen'), 12.88046587258031)
(('colonel', 'younghusband'), 12.719002449886192)
(('archive', 'foundation'), 12.513683541908687)
(('taj', 'mahal'), 12.489354212546948)
(('nur', 'jehan'), 12.414147868357773)
(('gutenberg', 'literary'), 12.207696990890346)
(('queen', 'victoria'), 12.179127838693574)
(('red', 'sandstone'), 12.10804474063209)
(('khyber', 'pass'), 12.073110950522707)
(('precious', 'stones'), 12.058719858217085)
(('burning', 'ghat'), 11.969363025684874)
(('mechanical', 'industries'), 11.942526840712562)
(('industrial', 'arts'), 11.88046587258031)
(('brass', 'bowl'), 11.772601839270 25)
(('saracenic', 'architecture'), 11.741094038011758)
(('arabian', 'sea'), 11.736075963245135)
(('grand', 'lama'), 11.666402435438322)
(('savings', 'banks'), 11.665686635353737)
(('nadir', 'shah'), 11.5985724394952)
```

Both the bigram lists obtained from raw_freq and PMI made sensible information. If there was a comparison between them, the PMI included words which gave more attention to the history of India.

c) I modified the stop word list for both the books because there were words which could be ignored and did not make great significance.
For Flight book – I created a new stop word list which contained words from the NLTK stop words list and to add to it I appended two more words to the list I created and ran the filter for bigrams using the updated list. The words which were added to my list were “much” and “made”

In [113]:
```
# create an empty list
wordsFiltered = []
# add the stopwords to this existing list
for w in stopwords:
        wordsFiltered.append(w)
#add custom stopwords
wordsFiltered.append('much')
wordsFiltered.append('made')
print(wordsFiltered)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your', 'yours', 'yourself', 'yourselves', 'he',
'him', 'his', 'himself', 'she', 'her', 'hers', 'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs',
 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'b
e', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'i
f', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ove
r', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'bot
h', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than',
 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'should', 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain',
'aren', 'couldn', 'didn', 'doesn', 'hadn', 'hasn', 'haven', 'isn', 'ma', 'mightn', 'mustn', 'needn', 'shan', 'should
n', 'wasn', 'weren', 'won', 'wouldn', 'much', 'made']
```

For Indian book – I created a new stop word list which contained words from the NLTK stop words list and to add to it I appended five more words to the list I created and ran the filter for bigrams using the updated list. The words which were added to my list were “much”, “made”, “may”, “many”, “even”.

In [50]:
```
# create an empty list
wordsFiltered = []
# add the stopwords to this existing list
for w in stopwords:
        wordsFiltered.append(w)
#add custom stopwords
wordsFiltered.append('much')
wordsFiltered.append('made')
wordsFiltered.append('may')
wordsFiltered.append('many')
wordsFiltered.append('even')
print(wordsFiltered)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your', 'yours', 'yourself', 'yourselves', 'he',
'him', 'his', 'himself', 'she', 'her', 'hers', 'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs',
 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'b
e', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'i
f', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ove
r', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'bot
h', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than',
 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'should', 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain',
'aren', 'couldn', 'didn', 'doesn', 'hadn', 'hasn', 'haven', 'isn', 'ma', 'mightn', 'mustn', 'needn', 'shan', 'should
n', 'wasn', 'weren', 'won', 'wouldn', 'much', 'made', 'may', 'many', 'even']
```

d) Yes, I ran trigram lists and have included them in part 3.

3. a) Since these books are of different genre, comparing them would be very difficult as one book is about the history of aero planes and the other is about Modern India. I could draw a questions from these books

- From the two books which we have based on the frequent words, bigrams and trigrams what can we infer from the books as to what it contains?

b)

- From the two books which we have based on the frequent words, bigrams and trigrams what can we infer from the books as to what it contains?
  For the Flight book –
  We can infer that this book mainly deals with aero planes and various machines which were used for manufacturing them when we consider words like “flight, motor, flying”.

```
('flight', 50)
('motor', 47)
('per', 44)
('miles', 43)
('could', 42)
('first', 39)
('flying', 39)
('surfaces', 39)
('time', 37)
('speed', 36)
('experiments', 35)
('works', 34)
('hour', 33)
('air', 32)
```

  This book also covers a lot of technical aspects of these machines which involve in what units the speed of the aero plane was measured, units used to measure how high it flies in the air when we consider words like “angle, degrees, pressure”.

```
('angle', 31)
('operator', 31)
('ground', 31)
('gutenberg', 30)
('power', 30)
('two', 29)
('us', 29)
('degrees', 29)
('pressure', 27)
```

Using the Bigram frequencies, we can easily conclude that this book mainly dealt with early history of aero planes. Bigrams like “gliding experiments, curved surfaces, first flight, flying problems” also explain how it was used initially. Also, several technical terms are used in the bigrams which indicate that this book contained detailed explanations of the aero planes and how they can fly.

```
(('per', 'cent'), 0.0004117344312918168)
(('per', 'second'), 0.0004117344312918168)
(('public', 'domain'), 0.0004117344312918168)
(('devil', 'hill'), 0.0003602676273803397)
(('full', 'project'), 0.0003602676273803397)
(('three', 'degrees'), 0.0003602676273803397)
(('curved', 'surfaces'), 0.00030880082346886257)
(('early', 'history'), 0.00030880082346886257)
(('first', 'flight'), 0.00030880082346886257)
(('flying', 'machines'), 0.00030880082346886257)
(('horizontal', 'position'), 0.00030880082346886257)
(('hundred', 'feet'), 0.00030880082346886257)
(("'s", 'body'), 0.0002573340195573855)
(('feet', 'per'), 0.0002573340195573855)
(('gutenberg-tm', 'work'), 0.0002573340195573855)
(('gutenberg-tm', 'works'), 0.0002573340195573855)
(('machine', 'would'), 0.0002573340195573855)
(('new', 'machine'), 0.0002573340195573855)
(('operator', "'s"), 0.0002573340195573855)
(('professor', 'langley'), 0.0002573340195573855)
(('would', 'develop'), 0.0002573340195573855)
(('copyright', 'holder'), 0.0002058672156459084)
(('first', 'trial'), 0.0002058672156459084)
(('flying', 'problem'), 0.0002058672156459084)
(('four', 'miles'), 0.0002058672156459084)
(('free', 'flight'), 0.0002058672156459084)
(('front', 'rudder'), 0.0002058672156459084)
(('gliding', 'experiments'), 0.0002058672156459084)
(('greater', 'angle'), 0.0002058672156459084)
(('gutenberg', 'ebook'), 0.0002058672156459084)
(('gutenberg-tm', 'trademark'), 0.0002058672156459084)
```

From the trigram frequencies we obtained we can infer that this book was one of the initial books which were written about flights by seeing trigrams like “early,history,of” and also trigrams like “as,a,kite” and “into,the,air” allows us to infer that this was the time when the author was trying to make an aero plane fly in the air which is nothing but the history of aero planes.

```
(('the', 'early', 'history'), 14.42725379619516)
(('head', 'resistance', 'of'), 14.210356260879365)
(('would', 'have', 'been'), 14.041751863154506)
(('angles', 'of', 'incidence'), 13.96593753321292)
(('the', 'united', 'states'), 13.886685414832453)
(('project', 'gutenberg-tm', 'works'), 13.476163396676107)
(('the', 'head', 'resistance'), 13.311776578775223)
(('angle', 'of', 'incidence'), 13.08213055071744)
(('center', 'of', 'pressure'), 13.018404953107055)
(('at', 'an', 'angle'), 12.766971204526051)
(('a', 'little', 'over'), 12.712523420503675)
(('project', 'gutenberg-tm', 'work'), 12.656735642317926)
(('as', 'a', 'kite'), 12.578442235055885)
(('through', 'the', 'air'), 11.896739079496378)
(('we', 'found', 'that'), 11.318362796356965)
(('part', 'of', 'this'), 11.255444150407904)
(('into', 'the', 'air'), 11.246681550553337)
```

For the Indian book –

From the frequency distribution table, we can infer that this book contained a lot of present day terms like government, city, religions and so on. It also took a dive back into the past where we have words like British, native and so on.

```
('india', 734)
('one', 481)
('upon', 439)
('people', 292)
('great', 281)
('government', 265)
('every', 249)
('years', 241)
('native', 226)
('several', 224)
('men', 210)
('many', 200)
('city', 185)
('feet', 180)
('would', 168)
('time', 166)
('women', 164)
('two', 161)
('hindu', 160)
('bombay', 150)
('work', 150)
('british', 149)
('may', 146)
('country', 137)
```

From the bigram frequencies, we can infer that the author has described about various historical places around India like the kutab minar, taj mahal, khyber pass and so on. Also, there are bigrams which allow us to infer that there was lot of emphasis on culture and the history of the nation when we consider words like nautch dancers, nur jehan. Bigrams like queen Victoria, warren hastings allow us to infer that the book has parts where the independence of India was also discussed.

```
(('melting', 'snows'), 14.999110369078931)
(('kutab', 'minar'), 14.736075963245135)
(('warren', 'hastings'), 14.321038463966289)
(('prime', 'minister'), 14.151113462523977)
(('bullock', 'carts'), 13.835611636796049)
(('nana', 'sahib'), 13.584072869800085)
(('edward', 'vii'), 13.513683541908687)
(('funeral', 'pyres'), 13.513683541908687)
(('austin', 'de'), 13.473041557411342)
(('literary', 'archive'), 13.414147868357773)
(('nautch', 'dancers'), 13.291291120572238)
(('funeral', 'pyre'), 13.028256714738445)
(('aga', 'khan'), 12.92872104118753)
(('fellow', 'countrymen'), 12.88046587258031)
(('colonel', 'younghusband'), 12.719002449886192)
(('archive', 'foundation'), 12.513683541908687)
(('taj', 'mahal'), 12.489354212546948)
(('nur', 'jehan'), 12.414147868357773)
(('gutenberg', 'literary'), 12.207696990890346)
(('queen', 'victoria'), 12.179127838693574)
(('red', 'sandstone'), 12.10804474063209)
(('khyber', 'pass'), 12.073110950522707)
(('[', 'illustration'), 12.0609156535524)
(('precious', 'stones'), 12.058719858217085)
(('burning', 'ghat'), 11.969363025684874)
(('mechanical', 'industries'), 11.942526840712562)
(('industrial', 'arts'), 11.88046587258031)
(('brass', 'bowl'), 11.77260183927025)
(('saracenic', 'architecture'), 11.741094038011758)
(('arabian', 'sea'), 11.736075963245135)
(('grand', 'lama'), 11.666402435438322)
(('savings', 'banks'), 11.665686635353737)
(('nadir', 'shah'), 11.5985724394952)
```

From the trigram frequencies which I obtained, we can infer that the book has parts which talks about the history of India when we look at words like “the,taj,mahal”,”. The author also speaks a lot about India from the time

the British came in. Words like “east,india,company”, “hundred,years,ago”,”the,municipal,authorities” allow us to infer that this book also gives us insight into the how India fought independence and how Indian is now.

```
(('hundred', 'years', 'ago'), 17.488873262368905)
(('project', 'gutenberg-tm', 'work'), 17.484887914417726)
(('a', 'previous', 'chapter'), 17.220965422965932)
(('set', 'forth', 'in'), 17.018271627881955)
(('you', 'can', 'imagine'), 16.723493165370552)
(('if', 'you', 'want'), 16.655634882059722)
(('i', 'have', 'already'), 16.404943424682276)
(('largely', 'due', 'to'), 16.345100693630773)
(('around', 'his', 'neck'), 16.344673406392047)
(('point', 'of', 'view'), 16.1814370587793)
(('the', 'taj', 'mahal'), 15.97916112332075)
(('some', 'years', 'ago'), 15.873696148600743)
(('the', 'khyber', 'pass'), 15.867772442824922)
(('come', 'in', 'contact'), 15.585312220605847)
(('arts', 'and', 'industries'), 15.56958838451871)
(('lord', 'curzon', 'has'), 15.564984799091711)
(('may', 'be', 'found'), 15.545361773444046)
(('may', 'be', 'seen'), 15.535093437990216)
(('the', 'arabian', 'sea'), 15.530737455547353)
(('the', 'burning', 'ghat'), 15.500990112153303)
(('the', 'grand', 'lama'), 15.46106392774055)
(('of', 'red', 'sandstone'), 15.438270125444767)
(('the', 'middle', 'ages'), 15.308345034210905)
(('the', 'municipal', 'authorities'), 15.29871633301314)
(('millions', 'of', 'dollars'), 15.296914276199235)
(('wives', 'and', 'daughters'), 15.295333191571416)
(('read', 'and', 'write'), 15.260567773410738)
```

Also, for reference I have included the function where I applied regular expressions to remove non-alphabetical characters

In [78]:
```
# using regular expression we remove the non alphabetical characters

import re
pattern = re.compile('^[^a-z]+$')
```

In [79]:
```
# function to return the list of words which
# do not include non alphabetical characters

def alpha_filter(w):
    pattern = re.compile('^[^a-z]+$')
    if(pattern.match(w)):
        return True
    else:
        return False
```