Week1_Assignment

August 2, 2019

1 Exercise 1 of Week 1 Assignement

Consider the text available in this website http://shakespeare.mit.edu/allswell/full.html- to answer the following questions - (a) Find the frequency the word "BERTRAM" (all in caps, ignore font types such as bold, italics, etc.) that appears at the start of the sentence? - (b) Is the ratio of the frequency of

$$\frac{BERTRAN}{Bertran} > 0$$

- (c) What is the average sentence length of this document? - (d) What is the vocabulary (unique set of words) size? Ignore alpha- numeric and numeric terms, if available - (e) Find the total count of the words that end with the exclamation mark(!)

```
In [1]: import re
        import urllib.request
        from bs4 import BeautifulSoup
        import nltk
        from nltk.tokenize import word_tokenize # for spliting the text into words
        from nltk.tokenize import sent_tokenize # for splitting the text into sentences
        from nltk.tokenize import RegexpTokenizer # for extracting text ending with !
        from nltk.corpus import inaugural # for average sentence length
```

1.1 Download the Text

```
In [2]: url = 'http://shakespeare.mit.edu/allswell/full.html'
        html = urllib.request.urlopen(url).read().decode('utf-8')
        soup = BeautifulSoup(html)
        raw = BeautifulSoup(html, 'html.parser').get_text()
        raw[:20]

        #with open("text.txt", "w") as file:
        #    file.write(raw)

Out[2]: "\n\n\nAll's Well That E"
```

1.2 Preprocess the Text

1.2.1 Split the text into tokens

```
In [3]: # word tokens
        word_tokens = word_tokenize(raw)
        sentence_tokens = sent_tokenize(raw)

In [4]: len(word_tokens)

Out[4]: 29364

In [5]: len(sentence_tokens)

Out[5]: 1442
```

1.2.2 Use NLTK Text to have only valid tokens

```
In [6]: text = nltk.Text(word_tokens)

In [7]: # (a) Find the frequency the word "BERTRAM" (all in caps, ignore font types such as bold, italics, etc.) that appears at the start of
        text.count('BERTRAM')

Out[7]: 127

In [8]: # (b) Is the ratio of the frequency of BERTRAN Bertram
        text.count('BERTRAM') / text.count('Bertram')

Out[8]: 18.142857142857142
```

In [9]:
```
# (c) What is the average sentence length of this document?
total_len = 0
total_sent = 0
for sent in sentence_tokens:
    #print(sent)
    total_len += len(sent)
    total_sent += 1

print("total_len = {0}, total_sent = {1} , Avg Sent Len = {2}".format(total_len, total_sent, total_len / total_sent))
```

```
total_len = 129890, total_sent = 1442 , Avg Sent Len = 90.07628294036061
```

In [10]:
```
# (d) What is the vocabulary (unique set of words) size? Ignore alpha- numeric and numeric terms, if available
vocab = [v.lower() for v in text.vocab() if v.isalpha()]
vocab = set(vocab)
print(len(vocab))
#print(vocab)
```

```
3299
```

In [11]:
```
# This method doesn't work as expected

# (e) Find the total count of the words that end with the exclamation mark(!)
#TokenSearcher(raw).findall(r'<^[a-z][A-Z]+!>')
# print(len(TokenSearcher(raw).findall(r'[A-Z][a-z]\w*!')))
```

In [12]:
```
# (e) Find the total count of the words that end with the exclamation mark(!)
regex_tokenizer = RegexpTokenizer('\w+!')
regex_tokens = regex_tokenizer.tokenize(raw)
print(len(regex_tokens), regex_tokens)
```

```
131 ['tis!', 'living!', 'shape!', 'birthright!', 'head!', 'him!', 'you!', 'all!', 'queen!', 'monarch!', 'up!', 't!', 't!', 't!', 'well!', 'to
```

2 Exercise 2 of Week 1 Assignement - Construct the binary incidence matrix

Construct the binary incidence matrix using the features extracted from the corpus. The corpus (271 text documents) is available at https://github.com/Ramaseshanr/anlp/blob/master/corpus/phy_corpus.txt. It contains contains questions from Kinematics class of physics problems sourced from the Internet - In this assignment, you need to develop a python program that uses the knowledge related to Kinematics and build a table similar to the one shown below for all the documents in the corpus. - The program should be able to read each problem, capture the known values (such as speed=10m/s, time=5s) and fill the respective cells in the table. For example, if you find 10 m/s for document 1, fill the speed with value row for D1 as 1. - Please note that problems may or may not contain all nine terms listed. - The corpus may contain duplicate entries - You may use any NLTK or any equivalent APIs for this assignment

Terms	D1	D2	…	D271
Speed with value	1	0	…	0
Distance with value	0	0	…	1
Acceleration with value	0	0	…	0
Time with value	0	1	…	0

In [13]:
```
import urllib.request
from bs4 import BeautifulSoup
import nltk
from nltk.tokenize import RegexpTokenizer # to get Kinematics
from nltk.tokenize import sent_tokenize # to find number of sentences in this corpus
from nltk.tokenize import word_tokenize
import pandas as pd # for table creation
```

2.1 Download the corpus

In [14]:
```
url = 'https://raw.githubusercontent.com/Ramaseshanr/anlp/master/corpus/phy_corpus.txt'

# Download the corpus as text
html = urllib.request.urlopen(url).read().decode('utf-8')
soup = BeautifulSoup(html)
raw = BeautifulSoup(html, 'html.parser').get_text()
raw[:20]

# WRITE IT TO THE FILE if required
#with open("text.txt", "w", encoding="utf-8") as file:
#    file.write(raw)
```

Out[14]:
```
'An airplane accelera'
```

2.2 Preprocess the corpus

- Each Paragraph is considered as a separate document

In [15]:
```
# Extract paragraphs
# Since text has indices in front of the lines, we can't use sent_tokenize()
# So splitting teh corpus into paragraphs and then I will extract kinematics from each paragraph
paragraphs = [para for para in raw.split('\n') if para]
```

In [16]:
```
# Ensure we have right number of paragraphs that is found in the corpus
len(paragraphs)
```

Out[16]: 271

2.3 Find the various kind of units found in the given Corpus

In [17]:
```
# Create RegEx Tokenizer which extracts the required kinematics
# Looking for 'x.xx letter' | 'x.xx word' | 'x.xx word|word'
#regex_tokenizer = RegexpTokenizer('\d+\.\d* \w |\d+\.\d* [a-z]* |\d+\.\d* [a-z]*/[a-z][0-9]* |\w\.\d+ [a-z]*')
#pattern1 = '\d+.\d+\s[a-zA-Z]*\/[a-zA-Z]*[\d]*' # x{anything}xx X/X
#pattern1 = '\d+.[\da-zA-Z]+\s[a-zA-Z]*\/[a-zA-Z]*[\d]*' # x{anything}xx X/X
pattern1 = '\d+.[\da-zA-Z]+\s[a-zA-Z]*[\/a-zA-Z\d\^\-]{1,}' # x{anything}xx X/X
pattern2 = '\d+\s[a-zA-Z]*\/[a-zA-Z]*[\d]*' # x X/X
pattern3 = '\d+.\d+\s[a-zA-Z]+' # x{anything}xx X
pattern4 = '\d+\s[a-zA-Z]+' # x XX
pattern5 = '\.\d+\s[a-zA-Z]+' # .x XX
regex_tokenizer = RegexpTokenizer(pattern1 + '|' + pattern2 + '|' + pattern3 + '|' + pattern4 + '|' + pattern5)
para_index = 0
total_tokens = 0
para_tokens = []
final_tokens = []
for paragraph in paragraphs:
    regex_tokens = regex_tokenizer.tokenize(paragraph)
    para_tokens.insert(para_index, [])
    if len(regex_tokens):
        final_tokens.extend(regex_tokens)
        para_tokens[para_index].extend(regex_tokens)
        print(para_index+1, regex_tokens)
        total_tokens += len(regex_tokens)
    para_index = para_index + 1
print('Total tokens found: ', total_tokens)
#print(final_tokens)
```

```
1 ['3.20 m/s2', '32.8 s']
2 ['5.21 seconds', '110 m']
3 ['2.60 seconds']
4 ['18.5 m/s', '46.1 m/s', '2.47 seconds']
5 ['1.40 meters', '1.67 m/s2']
6 ['444 m/s', '1.83 seconds']
7 ['7.10 m/s', '35.4 m']
8 ['3 m/s2', '65 m/s']
9 ['22.4 m/s', '2.55 s']
10 ['2.62 m']
11 ['1.29 m']
12 ['521 m/s', '0.840 m']
13 ['6.25 s']
14 ['370 m above']
15 ['367 m/s', '0.0621 m']
16 ['3.41 s']
17 ['290 m in', '3.90 m/s2']
18 ['88.3 m/s', '1365 m to']
19 ['112 m/s', '398 m']
20 ['1 m/s', '2.23 mi/hr', '91.5 m']
21 ['0.5 km', '1 hour later']
22 ['12 m/sec', '36 seconds']
23 ['2 m/s', '12 s he/she']
24 ['50 km traveling', '10 km/hr']
25 ['12 m/s', '3.00 minutes']
26 ['25 min at', '12 m/s']
27 ['3250 m/s', '10 m/s2', '215 km']
28 ['0.6 m/s2', '55 mi/h', '60 mi/h']
29 ['23.7 km/h', '0.92 m/s2', '3.6 s']
30 ['30 degree hill', '3.30 m/s2', '110 m']
31 ['48 m/s', '12 m/s', '5 s']
32 ['24 m/s', '315 m']
33 ['50 km/hr', '90 km/hr', '15 seconds']
34 ['9000 meters in', '12.12 seconds']
```

```
35 ['528 meters in', '4 seconds']
36 ['3 minutes running', '6 m/s']
37 ['450 km at', '120 m/s']
38 ['1000 m in', '20 minutes']
39 ['15 seconds', '0.8 m/sec', '7 m/sec']
40 ['1.0 km/s', '1.8 km/s', '0.03 seconds']
41 ['1700 m/s', '25 seconds']
42 ['12 m/sec', '36 seconds']
43 ['60 m/s', '8.0 sec']
44 ['6.00 m/s2', '4.10 seconds']
45 ['6 meter from']
46 ['189 m/s', '20 m off']
47 ['0.8 m/s', '10.3 seconds']
48 ['3 seconds to']
49 ['1 second it', '2 seconds']
50 ['4 seconds', '3 seconds and']
51 ['7 m/s', '20 seconds']
52 ['115 meters high']
53 ['110 m/s']
55 ['29 m/s', '10 m/s']
56 ['64 m in', '4 seconds']
57 ['0)above the', '16 m/s', '10 m above']
58 ['0.80 second']
59 ['12.0 m/s', '70.0 m']
60 ['10 m/s2', '4 seconds']
61 ['6 m/s', '2 m/s2', '7 m from']
62 ['16 meters from']
63 ['30 m/s']
64 ['1 second', '18 m/s', '3 seconds later']
65 ['8 m/s2', '10 seconds', '10 seconds', '10 seconds']
66 ['20 km/h', '8 m/s2', '10 seconds', '10 seconds', '10 seconds']
67 ['0 to 72', '11.5 seconds', '72 km/h']
68 ['40 m/s', '100 m']
69 ['5 m/s', '20 m/s', '10 seconds', '10 seconds']
70 ['350 km/h', '600 meters', '600 meters']
71 ['360 km/h', '10 m/s2']
72 ['8 seconds after', '340 m/s']
73 ['10 m']
74 ['10 m/s', '3.0 seconds']
75 ['239 m high', '3.7 m/s^2']
76 ['1.40 m', '1.67 m/s2']
77 ['2.6 seconds']
78 ['1.29 m']
79 ['420 m above']
80 ['2 m/s']
81 ['6 seconds of']
82 ['9.0 seconds']
83 ['2008 the Peachtree', '215 meters']
84 ['15 seconds to', '2nd second', '5th second']
85 ['3,000 meters']
86 ['132 m high', '3.0 seconds', '3.0 seconds', '1.63 m/s2']
87 ['0.45 meters']
88 ['6 seconds of']
89 ['9.0 seconds']
90 ['2008 the Peachtree', '215 meters']
91 ['1.85 seconds', '1.10 seconds']
92 ['3.34 seconds']
93 ['300.5 m']
94 ['98.5 m/s']
95 ['5.65 seconds']
96 ['4.0 s', '4.0 s']
97 ['1 instead threw', '8.0 m/s', '4.0 s', '8.0 m/s']
98 ['1 instead threw', '8.0 m/s', '4.0 s', '8.0 m/s']
99 ['8.0 m/s', '3.0 m/s']
100 ['20.0 m', '10.0 m/s', '0 m/s']
101 ['2.5 seconds']
102 ['1.75 seconds']
103 ['2.857 seconds']
104 ['9.2 meters']
105 ['1000 m']
106 ['1 meter tall']
107 ['9.2 meters']
108 ['1000 m']
109 ['1 meter tall']
110 ['9.5 seconds']
111 ['4 seconds to']
```

112 ['0.6 seconds']
113 ['11 seconds later']
114 ['9.5 seconds']
115 ['0.6 seconds']
116 ['11 seconds later']
117 ['4 seconds to']
118 ['18 meters']
119 ['9000 meters when']
120 ['1250 feet high', '1 foot', '0.305 meters']
121 ['0.5 seconds', '2.00 m/s/s', '2.20 m/s/s']
122 ['4.0 meters/second', '1 second', '1.5 m/s/s']
123 ['48 m/s', '12 m/s', '5 s']
124 ['24 m/s', '315 m']
125 ['17.12 U', '24.55 Columbia', '25.71 U', '27.37 Wichita', '30.34 U', '2000 m long']
126 ['45 miles straight', '75 mph', '85 mph', '20 minutes she']
127 ['10^6 m/s', '10^14 m/s2', '10^6 ms^-1']
128 ['2.0 m/s', '2 until it', '20 m/s', '20 s at', '5.0 s']
129 ['95 km/h', '130 km then', '65 km/h', '3 hours and', '20 min']
130 ['89.5 km/h', '22 min rest', '77.8 km/h']
131 ['0 to t', '4.21 min', '4.21 min', '8.42 min', '1.91 m/s', '1.00 min', '5.21 min']
132 ['0.684 km', '0.486 km', '3.56 km', '61.7 degrees', '1.124 h']
133 ['40 miles at', '80 mi/h', '40 mi/h']
134 ['100-mile race', '50 mi/hr', '100 mi/hr']
135 ['1.34 m/s', '6.44 km', '2.68 m/s', '0.447 m/s']
136 ['6 mi/h', '5 mi', '5 mi in', '10 miles is', '10.8 mi/hr']
137 ['1.50 km', '1.10 min']
138 ['3250 m/s', '10 m/s2', '215 km']
139 ['60 km/h', '2.0 s', '30 m and', '15 m wide', '6.4 m/s', '60 km/h', '70 km/h', '3.0 s']
140 ['0.6 m/s2', '55 mi/h', '60 mi/h']
141 ['23.7 km/h', '0.92 m/s', '2 for 3', '.6 s']
142 ['112 m/s', '20.0 s']
143 ['30 degree hill', '3.30 m/s', '110 m']
144 ['0.30 km', '5.0 m/s', '33.0 m/s']
145 ['24 m/s', '315 m']
146 ['50 km/hr', '90 km/hr', '15 seconds']
147 ['5.0 m/s', '4.0 seconds', '6.0 m/s']
148 ['40 km/h', '13 m ahead', '8.0 m/s/s', '0.25 s']
149 ['70 km/hr', '120 m']
150 ['20 m/s', '3.5 s']
151 ['20 seconds its', '108 kmh']
152 ['20 m/s', '3.5 s']
153 ['360 km/h', '24 seconds']
154 ['1500 kilograms reaches', '15 metersvper second', '5 seconds after']
155 ['0 to 60', '15 seconds']
156 ['0 km/h', '100 km/h', '5 seconds', '5 m/s']
157 ['150 km/h', '50 m from']
158 ['3.20 m/s2', '32.8 s']
159 ['18.5 m/s', '46.1 m/s', '2.47 seconds']
160 ['3.30 m/s2', '88.0 m/s']
161 ['8-km trip', '22.6 m/s', '16.8 m/s']
162 ['25.0 m/s', '1.0 m/s2']
163 ['25.0 m/s', '2.0 m/s2', '5 seconds', '10 more seconds', '15 seconds']
164 ['0.5 mile/minute', '10 minutes', '.25 mile', '2 for 2', '12 minutes of']
165 ['100 km at', '133 km/h']
166 ['8 m/s', '0 if it', '3.0 seconds']
167 ['9.58 s', '100-m dash']
168 ['80 days', '40,075 km']
169 ['3.20 m/s2', '32.8 s']
170 ['5.21 seconds', '110 m']
171 ['420 m above']
172 ['2.6 seconds']
173 ['120 cm east', '15 seconds', '120 cm west', '24 more seconds']
174 ['380 km in', '60 km/hr']
175 ['521 m/s', '0.840 m']
176 ['367 m/s', '0.0621 m']
177 ['88.3 m/s', '1365 m to']
178 ['3.41 s']
179 ['1.29 m']
180 ['1.40 m', '1.67 m/s2']
181 ['0.926 m/s2', '15 seconds with', '15 m/s']
182 ['2.5 meters']
183 ['2.8 m']
184 ['23.7 km/h', '0.92 m/s2', '3.6 s']
185 ['4.30 m/s', '3.0 m/s2', '5.0 s']
186 ['5.0 s', '1.5 m/s2']
187 ['15.0 m/s', '2.0 m/s2', '10.0 m/s']

188 ['2.3 m/s', '55 m']
189 ['120 km/hr', '240 m']
190 ['32 m/s', '3.0 m/s', '2 for 9']
191 ['7.0 m/s', '0.80 m/s2', '245 m', '125 m', '67 m']
192 ['15 m/s', '2 m/s2', '2.5 seconds']
193 ['5 seconds with', '1.5 m/s2']
194 ['4.30 m/s', '3 m/s2', '5 seconds']
195 ['23.7 km/hr', '0.92 m/s2', '3.6 sec']
196 ['90 km/hr', '36 km/hr', '5 s']
197 ['90 km/hr', '10 seconds to', '36 km/hr', '15 s']
198 ['25 m/s', '5 m/s', '4 s']
199 ['2 m/s', '18 m/s', '8 s']
200 ['3 m/s', '5 m/s', '6 s', '30 m/s']
201 ['500 m due', '300 s and', '400 m in', '320 s in', '500 m', '400 m and']
202 ['2 towns 60', '2 hours']
203 ['2 meters/second', '4 minutes']
204 ['6 m/s', '14 m/s2', '7e6 m/s']
205 ['0.210 s', '1.35 meters']
206 ['6.5 seconds']
207 ['3 seconds before', '3 second drop']
208 ['7.0 m', '2.00 m/s']
209 ['2.0 m/s', '9.8 m/s', '2.5 m']
210 ['4.6 m/s']
211 ['460 m/s']
212 ['7 seconds after', '50 m/s']
213 ['24 m above']
214 ['1.50 m', '19.6 m/s']
215 ['2 m/s', '50 m above']
216 ['5.00 m/s', '105 m above']
217 ['9.8 m/s']
218 ['30 m/s', '4.0 seconds', '10 m/s']
219 ['12 m/s', '40 degrees above']
220 ['12 m/s', '30.0 m']
221 ['54 m high', '130 m from']
222 ['22.2 m/s', '36 m from']
223 ['6.5 m', '4.0 m/s']
224 ['215 km/h', '155 km/h', '78.0 m']
225 ['25 m/s']
227 ['2.5 m/s', '0.75 m/s', '4.0 s']
228 ['30.0 m']
229 ['10 m', '15 degrees']
230 ['38.9 m', '3.05 m', '20.4 m/s']
231 ['30 m/s', '2 near Earth']
232 ['6.8 m/s', '2 m away']
233 ['5.0 kg', '10.1 m', '8.6 m/s']
234 ['48 m above', '24 m/s']
235 ['125 m above', '65.0 m/s']
237 ['3.05 m', '2 m above', '10 m from', '45 degree angle']
238 ['8.0 m', '9.0 m']
239 ['15.0 m/s', '30 degrees above', '2.0 seconds']
240 ['25 m/s']
241 ['26 m away', '5 m/s']
242 ['20.2 m', '10 degrees above', '3.0 seconds']
243 ['2.00 m', '.55 m', '32 degrees']
244 ['301.5 m', '25 degree to']
245 ['3.00 kg', '176.4 m', '12.0 N']
246 ['60 degrees with', '5 seconds later']
247 ['30 m away', '5 m above', '3 seconds after']
248 ['11 meters']
249 ['32.5 m/s']
250 ['10 m']
252 ['41.8 m', '42.7 m/s', '33.0 degrees']
254 ['120 m/s', '55 m/s']
255 ['35 m down', '3.06 m/s', '45 m high']
256 ['00-kg rock', '5.00 m/s']
257 ['58 m and', '3.41 s']
258 ['20 meters and', '10 seconds to']
259 ['5 m/s', '10 m above']
260 ['4.9 m/s', '2 s', '9.8 m/s2']
261 ['10 m/s2']
262 ['8 s in', '50 m']
263 ['75 m tall', '150 m away']
264 ['2.64 m/s', '1.48 s']
265 ['54 m high', '130 m from']
266 ['10 m/s']
267 ['20 m/s2', '1.0 min']

```
268 ['150000000 km', '3.0E8 m/s']
269 ['2.15 is', '15 m/s', '18 m']
270 ['0.0 m/s', '4.0 m/s', '4.0 s']
271 ['73.5 m/s', '2.2 s']
Total tokens found:  600
```

```
In [18]: def is_number_repl_dot_isdigit(s):
             '''
             replaces '.' with '' in s and
             returns the result of isdigit(s)
             https://stackoverflow.com/questions/354038/how-do-i-check-if-a-string-is-a-number-float
             '''
             return s.replace('.', '', 1).isdigit()
```

```
In [19]: def get_tokens(final_tokens):
             '''
             Accepts list having strings
             splits into tokens separated by spaces
             and returns it as a set
             '''
             ll = []
             [ll.extend(token.split(' ')) for token in final_tokens]
             return set(ll)
```

```
In [20]: final_tokens = get_tokens(final_tokens)
         final_tokens = [s for s in final_tokens if not is_number_repl_dot_isdigit(s)]
         sorted(final_tokens)
```

```
Out[20]: ['0)above',
          '00-kg',
          '100-m',
          '100-mile',
          '10^14',
          '10^6',
          '2nd',
          '3,000',
          '3.0E8',
          '40,075',
          '5th',
          '7e6',
          '8-km',
          'Columbia',
          'Earth',
          'N',
          'Peachtree',
          'U',
          'Wichita',
          'above',
          'after',
          'ahead',
          'and',
          'angle',
          'at',
          'away',
          'before',
          'cm',
          'dash',
          'days',
          'degree',
          'degrees',
          'down',
          'drop',
          'due',
          'east',
          'feet',
          'foot',
          'for',
          'from',
          'h',
          'he/she',
          'high',
          'hill',
          'hour',
          'hours',
          'if',
          'in',
```

```
 'instead',
 'is',
 'it',
 'its',
 'kg',
 'kilograms',
 'km',
 'km/h',
 'km/hr',
 'km/s',
 'kmh',
 'later',
 'long',
 'm',
 'm/s',
 'm/s/s',
 'm/s2',
 'm/s^2',
 'm/sec',
 'meter',
 'meters',
 'meters/second',
 'metersvper',
 'mi',
 'mi/h',
 'mi/hr',
 'mile',
 'mile/minute',
 'miles',
 'min',
 'minutes',
 'more',
 'mph',
 'ms^-1',
 'near',
 'of',
 'off',
 'race',
 'reaches',
 'rest',
 'rock',
 'running',
 's',
 'sec',
 'second',
 'seconds',
 'she',
 'straight',
 't',
 'tall',
 'the',
 'then',
 'threw',
 'to',
 'towns',
 'traveling',
 'trip',
 'until',
 'west',
 'when',
 'wide',
 'with']
```

2.4 Create the untils table based on above finding

In [21]:
```
speed_with_val = ['minutes', 'min', 'mi/hr', 'mi/h', 'mile/minute', 'metersvper', 'meters/second', 'm/sec', 'm/s', 'ms^-1', 'mph']
distance_with_val = ['miles', 'mile', 'meters', 'meter','m', 'feet', '-km', '-mile', '-m', 'km']
acceleration_with_val = ['km/h', 'km/hr', 'km/s', 'kmh', 'm/s2', 'm/s^2', 'm/s/s' ]
time_with_val = ['hours', 'hour', 'second', 'seconds', 'sec', 's', 'times', 'h']
```

2.5 Create the table and fill it

In [22]:
```
column_header = ['Speed with value', 'Distance with value', 'Acceleration with value', 'Time with value']
df = pd.DataFrame(columns=column_header)
```

In [23]:
```
# loop through each paragraph tokens
```

```
    para_index = 0
    for tokens in para_tokens:
        tt = get_tokens(tokens)
        #print(tt)
        # loop through each token
        c1 = 0
        c2 = 0
        c3 = 0
        c4 = 0
        if any(s in tt for s in speed_with_val):
            c1 = 1
        if any(s in tt for s in distance_with_val):
            c2 = 1
        if any(s in tt for s in acceleration_with_val):
            c3 = 1
        if any(s in tt for s in time_with_val):
            c4 = 1
        #print(para_index+1, c1, c2, c3, c4)
        df = df.append(pd.Series([c1, c2, c3, c4], index=df.columns), ignore_index=True)
        para_index += 1
```

In [24]:
```
df
```

Out[24]:
```
    Speed with value Distance with value Acceleration with value  \
0                  0                   0                       1
1                  0                   1                       0
2                  0                   0                       0
3                  1                   0                       0
4                  0                   1                       1
..               ...                 ...                     ...
266                1                   0                       1
267                1                   1                       0
268                1                   1                       0
269                1                   0                       0
270                1                   0                       0

    Time with value
0                 1
1                 1
2                 1
3                 1
4                 0
..              ...
266               0
267               0
268               0
269               1
270               1

[271 rows x 4 columns]
```

In [25]:
```
df.transpose()
```

Out[25]:
```
                         0    1    2    3    4    5    6    7    8    9    ... 261 262  \
Speed with value          0    0    0    1    0    1    1    1    1    0  ...    0   0
Distance with value       0    1    0    0    1    0    1    0    0    1  ...    1   1
Acceleration with value   1    0    0    0    1    0    0    1    0    0  ...    0   0
Time with value           1    1    1    1    0    1    0    0    1    0  ...    1   0

                         263 264 265 266 267 268 269 270
Speed with value          1    0    1    1    1    1    1    1
Distance with value       0    1    0    0    1    1    0    0
Acceleration with value   0    0    0    1    0    0    0    0
Time with value           1    0    0    0    0    0    1    1

[4 rows x 271 columns]
```

3 Exercise 3 of Week 1 Assignment

Write a program to find out whether Mandelbrot's approximation really provides a better fit than Zipf's empirical law. Use the same corpus for Zipf and Mandelbrot approximation

- Zipf Law
- https://www.cs.swarthmore.edu/~richardw/classes/cs65/f18/lab01.html for loglog
- https://www.researchgate.net/publication/221200132_Exploring_Regularity_in_Source_Code_Software_Science_and_Zipf's_Law

In [26]:
```python
from operator import itemgetter
import nltk
# from nltk.probability import FreqDist
from nltk.corpus import stopwords

import matplotlib.pyplot as plt
import math
```

In [27]:
```python
stop_words = set(stopwords.words('english'))

words = nltk.Text(nltk.corpus.gutenberg.words('austen-emma.txt'))
#words = nltk.Text(nltk.corpus.gutenberg.words('carroll-alice.txt'))

# convert to lower case and consider only words
words = [word.lower() for word in words]  #if word.isalpha()
# remove stop words
# words = [word for word in words if word not in stop_words]
```

In [28]:
```python
# =============================================================================
# fDist = FreqDist(words[:2000])
#
# print(fDist.B())
# print(fDist.Nr(1153))
#
# #for k,v in fDist.items():
#     #print(k, v)
#
# fDist.plot()
# =============================================================================
```

In [29]:
```python
# build word frequency as dictionary
frequency = {}
for word in words:
    count = frequency.get(word, 0)
    frequency[word] = count + 1

# sort and create a list
freq_list = sorted(frequency.items(), key=itemgetter(1), reverse=True)
#print(type(freq_list), freq_list)
```

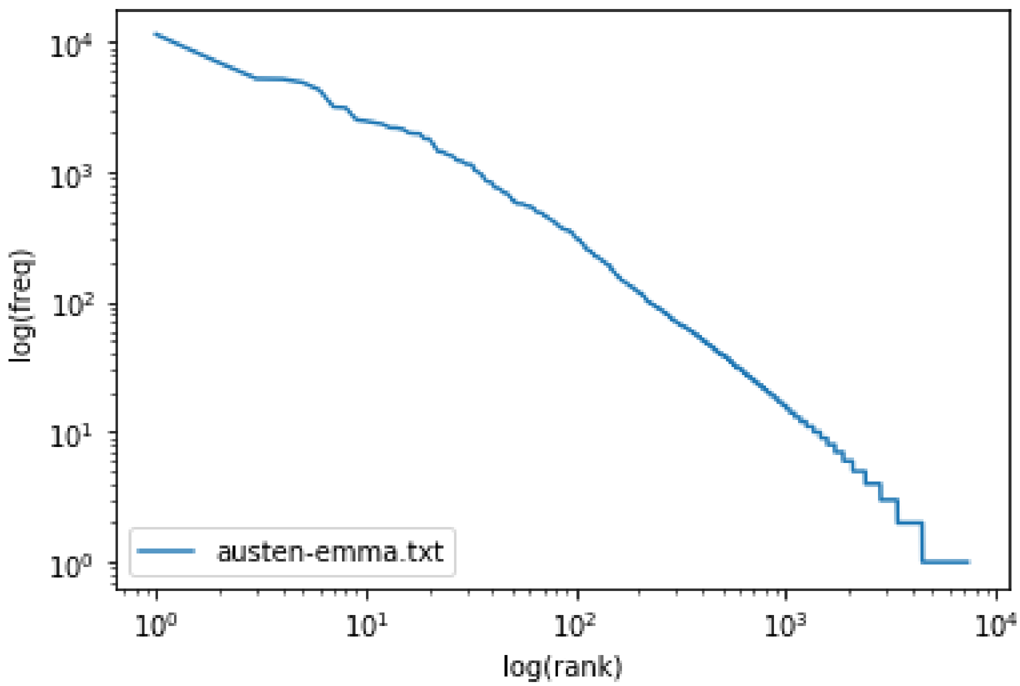
In [30]:
```python
# plot loglog graph of rank versus frequency
ranks = range(1, len(freq_list)+1) # x-axis: ranks
freqs = [freq for (word, freq) in freq_list] # y-axis: frequencies

print('50th Elem Freq: ', freqs[50])
print('150th Elem Freq: ', freqs[150])
print('150th*3 ==> ', freqs[150] * 3)

plt.loglog(ranks, freqs, label='austen-emma.txt')
plt.xlabel('log(rank)')
plt.ylabel('log(freq)')
plt.legend(loc='lower left')
```

```
50th Elem Freq:  599
150th Elem Freq:  173
150th*3 ==>  519
```

Out[30]:
```
<matplotlib.legend.Legend at 0x2d27382fcf8>
```

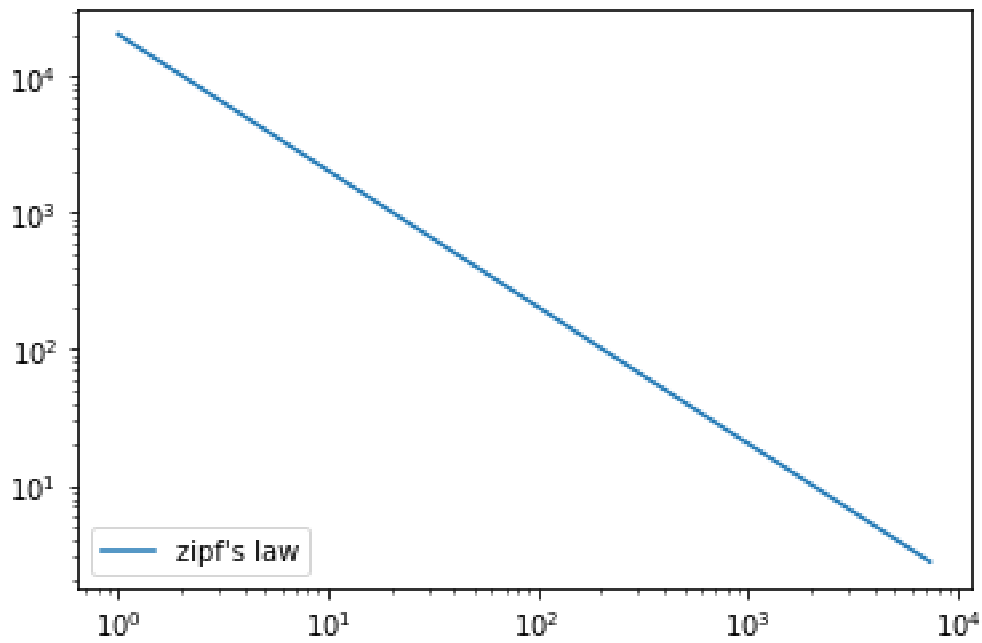



```
In [31]: # plot zip law's expected value
         # https://www.cs.swarthmore.edu/~richardw/classes/cs65/f18/lab01.html
         def H_approx(n):
             """
             Returns an approximate value of n-th harmonic number
             http://en.wikipedia.org/wiki/Harmonic-number
             """
             # Euler-Mascheroni constant
             gamma = 0.57721566490153286060651209008240243104215933593992
             return gamma + math.log(n) + 0.5/n - 1./(12*n**2) + 1./(120*n**4)

         T = len(words)
         print('T: ', T)
         vocab = set([word for (word, freq) in freq_list])
         n = len(vocab)
         print('n: ', n)
         k = T/H_approx(n)
         print('k: ', k)
         expected_freq = [k/r for r in ranks]
         plt.loglog(ranks, expected_freq, label="zipf's law")
         plt.legend(loc='lower left')

         plt.show()
```

```
T:  192427
n:  7344
k:  20300.51373217429
```



4 Exercise 4 of Week 1 Assignment

In [32]:
```
import nltk
from nltk.corpus import stopwords
import pandas as pd

stop_words = stopwords.words('english')
```

In [33]:
```
def getHeapLawValues(corpus_name):
    '''
    Accepts nltk.corpus.gutenberg corpus name
    Calculated vocabulary (M) and total words (T)
    Returns M, T, M/T as a list
    '''
    words = nltk.Text(nltk.corpus.gutenberg.words(corpus_name))
    # normalize the words
    words = [w.lower() for w in words if w.isalpha()]
    # remove stop words
    words = [w for w in words if w not in stop_words]

    M = len(set(words))
    T = len(words)

    print(corpus_name, ' M: ', M, ' T: ', T, 'Ratio: ', M/T)
    return [M, T, M/T]
```

In [34]:
```
column_headers = ['M', 'T', 'Ratio']
df = pd.DataFrame(columns=column_headers)

for corpus in nltk.corpus.gutenberg.fileids():
    df.loc[corpus] = getHeapLawValues(corpus)

print(df)
#df.plot()
df.Ratio.plot()
```

```
austen-emma.txt  M:  6948  T:  73149 Ratio:  0.09498421031046221
austen-persuasion.txt  M:  5606  T:  38337 Ratio:  0.1462294910921564
austen-sense.txt  M:  6148  T:  53986 Ratio:  0.11388137665320638
bible-kjv.txt  M:  12443  T:  374945 Ratio:  0.03318620064276094
blake-poems.txt  M:  1400  T:  3805 Ratio:  0.3679369250985545
bryant-stories.txt  M:  3688  T:  21718 Ratio:  0.16981305829266047
burgess-busterbrown.txt  M:  1382  T:  7582 Ratio:  0.18227380638353996
carroll-alice.txt  M:  2423  T:  12240 Ratio:  0.19795751633986927
chesterton-ball.txt  M:  8009  T:  39715 Ratio:  0.20166184061437745
chesterton-brown.txt  M:  7589  T:  35348 Ratio:  0.21469390064501528
```

```
chesterton-thursday.txt  M:  6159  T:  28328 Ratio:  0.21741739621575826
edgeworth-parents.txt  M:  8166  T:  78148 Ratio:  0.1044940369555203
melville-moby_dick.txt  M:  16802  T:  110459 Ratio:  0.15211073792085752
milton-paradise.txt  M:  8849  T:  45568 Ratio:  0.19419329353932585
shakespeare-caesar.txt  M:  2911  T:  11056 Ratio:  0.2632959479015919
shakespeare-hamlet.txt  M:  4590  T:  15898 Ratio:  0.288715561705875
shakespeare-macbeth.txt  M:  3340  T:  10078 Ratio:  0.3314149632863663
whitman-leaves.txt  M:  12189  T:  65080 Ratio:  0.18729256299938538
                               M         T     Ratio
austen-emma.txt           6948.0   73149.0  0.094984
austen-persuasion.txt     5606.0   38337.0  0.146229
austen-sense.txt          6148.0   53986.0  0.113881
bible-kjv.txt            12443.0  374945.0  0.033186
blake-poems.txt           1400.0    3805.0  0.367937
bryant-stories.txt        3688.0   21718.0  0.169813
burgess-busterbrown.txt   1382.0    7582.0  0.182274
carroll-alice.txt         2423.0   12240.0  0.197958
chesterton-ball.txt       8009.0   39715.0  0.201662
chesterton-brown.txt      7589.0   35348.0  0.214694
chesterton-thursday.txt   6159.0   28328.0  0.217417
edgeworth-parents.txt     8166.0   78148.0  0.104494
melville-moby_dick.txt   16802.0  110459.0  0.152111
milton-paradise.txt       8849.0   45568.0  0.194193
shakespeare-caesar.txt    2911.0   11056.0  0.263296
shakespeare-hamlet.txt    4590.0   15898.0  0.288716
shakespeare-macbeth.txt   3340.0   10078.0  0.331415
whitman-leaves.txt       12189.0   65080.0  0.187293
```

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x2d2717a22b0>