IST 664 – Natural Language Processing
HW 1 - Corpus Statistics and Python Programming
Kshitij Sankesara

1. Analysis of Wikipedia discussion forum:

We have been given a text file 'HW_WikipediaDiscussions.txt' for analyzing the Wikipedia discussion forum. I performed some steps for processing the data. I first removed the html tags which are in the text file using BeautifulSoup library as there are a lot of html tags in the file. Then using word_tokenize, I separated each word in a different index of the list. Then using the in built lower() function, I converted all the uppercase words to lowercase. We do this so that we don't count the same word as different entities. There are stopwords ('the', 'an', …) too in the file which I removed it using nltk library stopwords list. There are some characters in the text file which are actually not alphabets, so I removed them using the alpha_filter(). After all the processing steps, I found the top 50 words and their frequency using FreqDist class. Then by using collocations class, I found the bigrams. The bigrams were based on their raw frequency. I also found the Mutual Information score with a filter - frequency of minimum of 5.

2. Analysis of NPS Chat Corpus:

A large body of text is known as a Corpus. We are using the NPS Chat Corpus which is a part of nltk distribution. The NPS Chat Corpus was created by Jane Lin, Eric Forsyth and Craig Martell. There is a total of 15 xml documents and one readme and xsd file each. The text corpus contains over 10,000 posts as of now and there are various conventions which have been followed in the corpus. The purpose of the NPS Chat Corpus was to detect Internet predators automatically. The corpus is publicly available for research. To maintain the privacy of the user data, the author has made minor changes to the text file. The usernames have been changed to 'UserN' where N is a unique integer, which is used to identify a particular user.

The name of the file is such that it gives all the information. It gives us the total number of posts, the number of chat rooms from which the data was collected and the date on which the data was collected. The dataset which we have is obtained in the year of 2006. The chats in this are also age-specific. To maintain the privacy, the user names are changed to a generic name.

For this corpus, I have first used the lower(), alpha_filter() function and FreqDist class. The reason to use these functions is pretty much the same. To convert all the words to lowercase form, I have first used the lower() function. Then to remove tokens which are not alphabets, I have used the alpha_filter() function. We do this to get more precise analysis. Then using FreqDist, I have found the frequency of the top 50 words. Later, I found the Bigram frequency based on the raw frequency and then I calculated the Mutual Information score.

3. Comparison:

A) I compared the analysis results from both the questions 1 and 2. By comparing their frequency of words, I came to a conclusion that the language used in the Wikipedia Discussion is more formal as compared to the language used in NPS Chat Corpus. The Wikipedia Discussions is also grammatically correct in comparison with NPS Chat Corpus. The Wikipedia Discussions also had all the names of the author which was not the case with the NPS Chat Corpus. The NPS Chat Corpus had slang words, incorrect spellings and short form of words which was not there in the Wikipedia Discussions.

B) I used Beautiful Soup to remove HTML tags in Wikipedia Discussion whereas the data in the NPS Chat Corpus doesn't require that. This is one of the differences in processing the two datasets. I also removed the stop words in Wikipedia Discussion but didn't remove it from the NPC Chat Corpus as the data which we were left with after removing the stop words was very little.

C) In NPS Chat, some words with same meaning are counted as different words such as 'im' and 'I am'. Also, words like 'part' and 'join' appear a greater number of times than other bigrams. These words have no actual contribution to real chat data. There is a difference between bigrams using frequencies and bigrams using Mutual Information. The bigrams using frequency are just the words that appear together whereas the bigrams using Mutual Information is based on words which occur in a similar context.

4. Word and Name puzzle:

For generating the word and name puzzle, we first store the Frequency Distribution of the given letter in a variable called puzzle_letters. In the obligatory, we store the compulsory letter 'm' which should be there in all the words. Then in a variable called wordlist, we store the words

from nltk.corpus.words. This list will help us to check whether the word created is correct. The code consists of three important conditions. The first checks whether the length of each word is greater than or equal to 6. The second checks whether the compulsory letter exists in the word. The third and the final one checks whether each letter occurs only one time in the word.

Appendix :

1. Wikipedia Discussion forum

```
In [16]: from bs4 import BeautifulSoup

In [19]: Bsoup = BeautifulSoup(wikiDis, 'lxml')

In [21]: wd = Bsoup.get_text()

In [22]: print(wd[:100])
         ". This is just a curriculum vitae and WP:LINKFARM for a non-notable businessperson, and that's all

In [23]: wikiDistokens = nltk.word_tokenize(wd)

In [24]: wikiDiswords = [w.lower() for w in wikiDistokens]

In [25]: alphawikiDiswords = [w for w in wikiDiswords if not alphafilter(w)]

In [36]: stopwikiDiswords = [w for w in alphawikiDiswords if w not in stopwords]
```

Processing Steps

Top 50 words by Frequency:

```
In [38]: from nltk import FreqDist
         wikiDisdist = FreqDist(stopwikiDiswords)

In [39]: wikiDisitems = wikiDisdist.most_common(50)

In [74]: for item in wikiDisitems:
             print('(\'' + str(item[0]) + '\', ' + str(item[1]/len(wikiDiswords)) +')')
```

```
('article', 0.009396865054310331)
('wp', 0.008808487171781502)
('''s', 0.005351558484692799)
('sources', 0.005260253390290695)
('n't', 0.004583908449069086)
('notability', 0.004267187766766732)
('notable', 0.003701783424119723)
('coverage', 0.003097697138252898)
('new', 0.003069422012244504)
('please', 0.0026990964035512382)
('per', 0.0026451969445977375)
('add', 0.002602293367980835)
('one', 0.00256960025353363)
('comments', 0.0025573280634258204)
('thanks', 0.0025144244868089173)
('notice', 0.0024675938093575156)
('reliable', 0.002315418652020675)
```

```
('wikipedia', 0.0022595556426499254)
('articles', 0.002198096514590014)
('would', 0.0021138602016900084)
('gng', 0.002065753216467394)
('fails', 0.0018772523764114368)
('subject', 0.0017213464732818223)
('also', 0.0016842353703958057)
('page', 0.0016246416152322816)
('find', 0.0015705458012370563)
('see', 0.001466281274081105)
('significant', 0.0014544017940567452)
('list', 0.0014439949768453226)
('like', 0.0013574024034446173)
('independent', 0.0012908380442998573)
('enough', 0.0012872054760279456)
('even', 0.0012868127659444958)
('could', 0.0012123942051307376)
('source', 0.0011912860381453048)
('seems', 0.0011431790529226908)
('afd', 0.0010997845887014757)
('meet', 0.0010951702452209392)
('deletion', 0.0010870215109893536)
('think', 0.0010746511433606814)
('references', 0.0010596299826687223)
('delete', 0.000986193197063589)
('may', 0.0009464313011142854)
('news', 0.0009311156078597389)
('found', 0.0009270903295043773)
(''m', 0.0008862484808255866)
('non-notable', 0.0008740744682386393)
('nothing', 0.0008400068684993596)
('google', 0.0008345089273310608)
('two', 0.000833527152122436)
```

Top 50 bigrams by Frequency:

```
In [41]: wikiDisbigrams = list(nltk.bigrams(wikiDiswords))
```

```
In [45]: from nltk.collocations import *
         Bmeasures = nltk.collocations.BigramAssocMeasures()
         Bfinder = BigramCollocationFinder.from_words(stopwikiDiswords)
         Bscored = Bfinder.score_ngrams(Bmeasures.raw_freq)
```

```
In [47]: for Bscore in Bscored[:50]:
             print (Bscore)
```

```
(('please', 'add'), 0.005497985571914642)
(('add', 'new'), 0.005488940600997615)
(('new', 'comments'), 0.005485774861176656)
(('comments', 'notice'), 0.005484870364084953)
(('notice', 'thanks'), 0.005483513618447399)
(('wp', 'gng'), 0.0035022127390727143)
(('fails', 'wp'), 0.0031786289045160863)
(('reliable', 'sources'), 0.0028729088875205856)
(('per', 'wp'), 0.0019100717334031001)
(('significant', 'coverage'), 0.0016938969284861635)
```

```
(('non-admin', 'closure'), 0.0013447610510889353)
(('meet', 'wp'), 0.0013431781811784557)
(('thanks', 'please'), 0.0013189828839754094)
(('ca', "n't"), 0.0012638085613815469)
(('wikipedia', 'articles'), 0.00094904357346902)
(('gng', 'wp'), 0.0009180645480782037)
(('coverage', 'reliable'), 0.0009130898140738391)
(("n't", 'find'), 0.0007950529436066415)
(("n't", 'see'), 0.0007292507801852727)
(('article', "'s"), 0.0007265372889101646)
(('meets', 'wp'), 0.0007154571995368071)
(('wp', 'gng.'), 0.0007152310752638814)
(('per', 'nom'), 0.0006951060149734971)
(('find', 'sources'), 0.0006598306283970932)
(('establish', 'notability'), 0.00063495695837527)
(('books', 'scholar'), 0.0005956113348862041)
(('newspapers', 'books'), 0.0005924455950652447)
(('scholar', 'highbeam'), 0.0005886014824255084)
(('highbeam', 'jstor'), 0.0005881492338796571)
(('independent', 'reliable'), 0.0005770691445062994)
(("n't", 'think'), 0.000540210888019416)
(('notability', 'guidelines'), 0.0005399847637464903)
(('independent', 'sources'), 0.0005379496452901593)
(('reliable', 'source'), 0.000536140651106754)
(('comment', 'added'), 0.0005264173073709504)
(('unsigned', 'comment'), 0.0005223470704582883)
(('secondary', 'sources'), 0.0005203119520019574)
(('preceding', 'unsigned'), 0.0005200858277290317)
(('evidence', 'notability'), 0.0005013175130762014)
(('notable', 'enough'), 0.0004906896722486951)
(('talk', 'page'), 0.0004588061497661762)
(('pass', 'wp'), 0.0004513440487596292)
(('passes', 'wp'), 0.00044569094193648757)
(('looks', 'like'), 0.0004450125691177106)
(('jstor', 'free'), 0.0004287316214670626)
(('free', 'images'), 0.00042850549719413697)
(('images', 'wikipedia'), 0.00042850549719413697)
(('news', 'newspapers'), 0.00042782712437536)
(('wikipedia', 'library'), 0.0004262442544648803)
(("n't", 'seem'), 0.00042398301173562366)
```

Top 50 bigrams by Mutual Information Score:

```
In [48]: Bfinder2 = BigramCollocationFinder.from_words(stopwikiDiswords)
         Bfinder2.apply_freq_filter(5)

In [49]: Bscored = Bfinder2.score_ngrams(Bmeasures.pmi)
         for Bscore in Bscored[:50]:
             print (Bscore)
```

```
(('burr', 'steers'), 19.75445270527577)
(('helsingin', 'sanomat'), 19.75445270527577)
(('hemorrhagic', 'conjunctivitis'), 19.75445270527577)
(('inã©s', 'rodena'), 19.75445270527577)
(('khyber', 'pakhtunkhwa'), 19.75445270527577)
```

(('manadel', 'al-jamadi'), 19.75445270527577)
(('mys', '721tx'), 19.75445270527577)
(('pell', 'mell'), 19.75445270527577)
(('phnom', 'penh'), 19.75445270527577)
(('putroe', 'neng'), 19.75445270527577)
(('rot-weiãÿ', 'oberhausen'), 19.75445270527577)
(('schwã¤bisch', 'gmã¼nd'), 19.75445270527577)
(('sunanda', 'pushkar'), 19.75445270527577)
(('super-god', 'masterforce'), 19.75445270527577)
(('vis-ã', '-vis'), 19.75445270527577)
(('ashleigh', 'lollie'), 19.491418299441975)
(('beent', 'agged'), 19.491418299441975)
(('deletion/anshei', 'sfard'), 19.491418299441975)
(('deletion/beth', 'hamedrosh'), 19.491418299441975)
(('dudel250', 'chatprod'), 19.491418299441975)
(('energy-safety', 'energy-economy'), 19.491418299441975)
(('giro', "d'italia"), 19.491418299441975)
(('hamedrosh', 'hagodol-beth'), 19.491418299441975)
(('lorem', 'ipsum'), 19.491418299441975)
(('m.j.', 'ramanan'), 19.491418299441975)
(('margarita', 'martirena'), 19.491418299441975)
(('mong', 'kok'), 19.491418299441975)
(('movers', 'shakers'), 19.491418299441975)
(('rls=org.mozilla', 'en-us'), 19.491418299441975)
(('suhas', 'gopinath'), 19.491418299441975)
(('ulrike', 'ottinger'), 19.491418299441975)
(('vitalik', 'buterin'), 19.491418299441975)
(('xhulio', 'joka'), 19.491418299441975)
(('abdulhadi', 'najjar'), 19.269025878105527)
(('aqueduct', 'racetrack'), 19.269025878105527)
(('chal', 'jhoothey'), 19.269025878105527)
(('charles_manson', 'tate_murders'), 19.269025878105527)
(('deletion/tessa', 'campanelli'), 19.269025878105527)
(('diante', 'trono'), 19.269025878105527)
(('guo', 'dongli'), 19.269025878105527)
(('hidy', 'ochiai'), 19.269025878105527)
(('marlene', 'dietrich'), 19.269025878105527)
(('mushtaq', 'pahalgami'), 19.269025878105527)
(('officeâ€\x9dwp', 'polititon'), 19.269025878105527)
(('option=com_content', 'view=article'), 19.269025878105527)
(('polititon', 'states-'), 19.269025878105527)
(('rowman', 'littlefield'), 19.269025878105527)
(('sadman', 'sakibzz'), 19.269025878105527)
(('satish', 'rajwade'), 19.269025878105527)
(('sebalu', 'lule'), 19.269025878105527)

2. NPS Chat Corpus:

```
In [77]: from nltk.corpus import nps_chat

In [78]: Npschat = nltk.Text(nps_chat.words())

In [79]: Npswords = [w.lower() for w in Npschat]

In [80]: Alphanpswords = [w for w in Npswords if not alphafilter(w)]

In [84]: Npsdist = FreqDist(Alphanpswords)

In [85]: Npsitems = Npsdist.most_common(50)
```

Processing Steps

Top 50 words by Frequency –

```
In [87]: for item in Npsitems:
             print(item[0], item[1]/len(Npswords))
```

```
i 0.027193956898467007
part 0.02270606531881804
join 0.022683848033770274
lol 0.018262608309264607
you 0.015241057542768274
to 0.014774494556765163
the 0.014663408131526327
hi 0.014574538991335258
a 0.012886025327704954
me 0.009508998000444345
is 0.008442568318151522
in 0.008087091757387248
and 0.007931570762052876
it 0.007887136191957344
action 0.007709397911575206
hey 0.0064874472339480115
that 0.0063097089535658745
my 0.0057542768273716955
of 0.004598978004887803
u 0.004532326149744501
what 0.0044656742946011995
's 0.004332370584314597
on 0.004199066874027994
for 0.004199066874027994
here 0.004110197733836925
are 0.004021328593645857
do 0.004021328593645857
no 0.004021328593645857
not 0.003976894023550322
```

```
have 0.003799155743168185
all 0.0037325038880248835
up 0.0035769828926905133
like 0.003554765607642746
with 0.003421461897356143
im 0.0033103754721173074
pm 0.0033103754721173074
chat 0.0032437236169740057
n't 0.0031992890468784713
so 0.0031770717618307045
your 0.0031548544767829373
was 0.0031548544767829373
how 0.0030659853365918683
'm 0.0029548989113530326
good 0.0029326816263052654
any 0.002866029771161964
lmao 0.002843812486114197
just 0.002843812486114197
too 0.0028215952010664297
there 0.0026882914907798267
u7 0.0026438569206842922
```

Top 50 bigrams by Frequency –

```
In [92]: Bfinder3 = BigramCollocationFinder.from_words(Alphanpswords)
         Scored3 = Bfinder3.score_ngrams(Bmeasures.raw_freq)
```

```
In [93]: for bscore in Scored3[:50]:
             print (bscore)
```

```
(('part', 'join'), 0.005006064046654452)
(('join', 'part'), 0.003509405723427864)
(('i', "'m"), 0.00343199236188166 1)
(('part', 'part'), 0.0030449255541506464)
(('join', 'join'), 0.0027868810156633033)
(('pm', 'me'), 0.002374009754083555)
(('in', 'the'), 0.0019869429463525404)
(('i', 'am'), 0.0019095295848063376)
(('are', 'you'), 0.0017288984078651975)
(('wanna', 'chat'), 0.0015482672309240575)
(('it', "'s"), 0.0014966583232265888)
(('i', 'have'), 0.001419244961680386)
(('join', 'hi'), 0.0013418316001341832)
(('do', "n't"), 0.0013160271462854488)
(('lol', 'i'), 0.0013160271462854488)
(('join', 'i'), 0.0012902226924367147)
(('i', 'was'), 0.001238613784739246)
(('lol', 'lol'), 0.0012128093308905116)
(('part', 'i'), 0.0012128093308905116)
(('to', 'chat'), 0.0011870048770417775)
(('lol', 'part'), 0.0011095915154955744)
(('lol', 'hi'), 0.0010837870616468403)
(('lol', 'join'), 0.0010837870616468403)
(('want', 'to'), 0.0010837870616468403)
```

```
(('on', 'the'), 0.0010321781539493716)
(('part', 'hi'), 0.0010321781539493716)
(('how', 'are'), 0.0010063737001006373)
(('i', 'know'), 0.0010063737001006373)
(('part', 'lol'), 0.0010063737001006373)
(('join', 'action'), 0.000877351430856966)
(('join', 'lol'), 0.000877351430856966)
(('action', 'is'), 0.0008515469770082316)
(('i', 'do'), 0.0008257425231594974)
(('i', 'dont'), 0.0008257425231594974)
(('in', 'here'), 0.0008257425231594974)
(('have', 'a'), 0.000799938069310763)
(('you', "'re"), 0.0007741336154620288)
(('do', 'you'), 0.0007483291616132944)
(('i', 'think'), 0.0007483291616132944)
(('is', 'a'), 0.0007483291616132944)
(('#14-19teens', 'o'), 0.0007225247077645602)
(('i', 'just'), 0.0007225247077645602)
(('mode', '#14-19teens'), 0.0007225247077645602)
(('with', 'a'), 0.0007225247077645602)
(('i', 'like'), 0.0006967202539158258)
(('talk', 'to'), 0.0006967202539158258)
(('to', 'me'), 0.0006967202539158258)
(('i', 'can'), 0.0006709158000670916)
(('me', 'if'), 0.0006709158000670916)
(('part', 'action'), 0.0006709158000670916)
```

Top 50 bigrams by Mutual Information Score:

```
In [94]: Bfinder4 = BigramCollocationFinder.from_words(Alphanpswords)
         Bfinder4.apply_freq_filter(5)

In [95]: Scored4 = Bfinder4.score_ngrams(Bmeasures.pmi)
         for bscore in Scored4[:50]:
             print (bscore)
```

```
(('lez', 'gurls'), 12.242020378132263)
(('gently', 'kisses'), 12.072095376689951)
(('neck', 'compliments'), 11.809060970856155)
(('fingers', 'thru'), 11.51955435366117)
(('ice', 'cream'), 11.51955435366117)
(('played', 'times'), 11.278546254157373)
(('lime', 'player'), 11.154557536881923)
(('lasts', 'minutes'), 11.072095376689951)
(('cute.-ass', 'mp3'), 11.072095376689948)
(('hair', 'closes'), 10.89152313104813)
(('closes', 'their'), 10.731058458854882)
(('minutes', 'seconds'), 10.657057877411106)
(('talkin', 'bout'), 10.51955435366117)
(('la', 'la'), 10.331127851966247)
(('mp3', 'player'), 10.30656063032697)
(('runs', 'their'), 10.146095958133726)
(('their', 'fingers'), 9.856589340938744)
(('song', 'lasts'), 9.849702955353502)
(('mode', '#40splus'), 9.8497029553535)
```

```
(('minutes', 'ago'), 9.750167281802586)
(('mode', '#14-19teens'), 9.750167281802586)
(('busy', 'busy'), 9.652057196352738)
(('#14-19teens', 'o'), 9.387597202417878)
(('their', 'eyes'), 9.31602095957604)
(('#40splus', 'o'), 9.224098470135)
(('u99', 'u99'), 9.13051206291527)
(('been', 'played'), 9.121004977170893)
(('f', 'c'), 8.946087491777844)
(('f', 'tx'), 8.9200922832449)
(('or', 'lez'), 8.71845842207525)
(('u122', 'u122'), 8.627310534017054)
(('at', 'least'), 8.598164188357536)
(('mode', '#talkcity_adults'), 8.569595036160766)
(('main', 'room'), 8.555519850949043)
(('an', 'hour'), 8.51138042221547)
(('talking', 'about'), 8.48713287596879)
(('m', 'canada'), 8.330328796259924)
(('last', 'night'), 8.279702723189951)
(('times', 'this'), 8.248798910763322)
(('#talkcity_adults', 'o'), 8.207024956776056)
(('females', 'want'), 8.09227325862758)
(('player', 'song'), 8.084168208990526)
(('wo', "n't"), 8.07209537668995)
(('last', 'seen'), 8.02146930361998)
(('bi', 'or'), 7.981492827909042)
(('long', 'time'), 7.956618159270009)
(('song', 'has'), 7.916817151212038)
(('player', 'this'), 7.861775787654075)
(('who', 'sang'), 7.836028018456426)
(('ca', "n't"), 7.831087277186153)
```

3. Word & Name puzzle:

```
In [65]: puzzle_letters = nltk.FreqDist('egbdafkjlmorcnst')

In [66]: obligatory = 'm'

In [67]: wordlist = nltk.corpus.words.words()

In [72]: [w for w in wordlist if len(w) >= 6
          and obligatory in w
          and nltk.FreqDist(w) <= puzzle_letters]

Out[72]: ['abdomen',
          'abelmosk',
          'acneform',
          'almond',
          'almoner',
          'almost',
          'ambler',
          'ambrose',
          'amelcorn',
          'amends',
          'amlong',
          'amober',
          'amongst',
          'amoret',
          'angeldom',
          'angstrom',
          'antdom',
          'armbone',
          'armlet',
```