Practical 4

Text Analytics: Beyond Frequencies

Otto Hermann
`otto.hermann@ucdconnect.ie`
`16203034`

COMP47600

University College Dublin
13th October 2017

1 General

My tweets were a subset of the many tweets coming out of this incident of "mans-plaining": someone made a joke about Barry Gibb, Take That, and covering songs . . . and then hordes of the grimiest of men emerged from their mother's basements for a purpose only they understand.

As usual, I've removed comments from my code to save space, but you can find full comments in the actual code in q1.py.

1a: remove stop words

Here is the code I used to remove stop words using the standard nltk library:

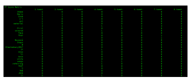
```python
def tokenize_text_file(file_name):
  with open(file_name, "r") as source:
    return nltk.word_tokenize(source.read())

def tokenize_text_file_remove_stop_words(file_name):
  tokens = tokenize_text_file(file_name)
  stop_words = set(stopwords.words('english'))
  return [t for t in tokens if t not in stop_words]
```

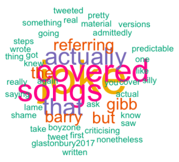
Because I was using tweets, I didn't turn terms into all lower case or conduct spelling checks as I wanted to capture as much of the original sentiment/tone as possible e.g. all caps of word-X is distinct from the normal spelling of word-X.

1b: compute TD scores

Per direction received from the Professor, I only implemented boolean versions of TF scores, but modifying the code to accommodate other methods of calculations would be straightforward.



(a) TF Score Matrix (selection)



(b) TF Score Wordcloud

Figure 1: TF Score Outputs

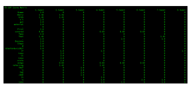
As the text represents an exchange of opinions on Twitter, there are a lot of peculiar elements to the text; however, both the matrix and the wordcloud identify the core elements well i.e. that this exchange is about a joke related to covering a song by Barry Gibb.

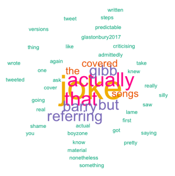
Below is the wordcloud command I used in R; the Python code used to generate the input text (and all other code for Q1) based on the TF Matrix values can be found in q1.py.

```
wordcloud("Shame Barry Barry Gibb Gibb n't got material . . . . . . First covered
     covered covered covered Take That That , , , , , Boyzone Steps # Glastonbury2017
     If If If knew 'll know wrote songs songs songs songs referring referring THAT 'S
     THE JOKE It 's 's 's 's joke joke joke joke really ? actually actually actually
     would would would No real You 'd 'd criticising actual cover versions But But
     saying written Again : The Joke Admittedly pretty lame predictable one
     nonetheless tweet silly thing I I tweeted something like saw ask going",
     colors=brewer.pal(6, "Dark2"),random.order=FALSE, rot.per=0, scale=c(4, .0625),
     min.freq=1)
```

1c: compute TD-IDF scores

Per direction received from the Professor, I only implemented boolean versions of TF-IDF scores, but modifying the code to accommodate other methods of calculations would be straightforward.


(a) TF-IDF Score Matrix (selection)



(b) TF Score Wordcloud

Figure 2: TF Score Outputs

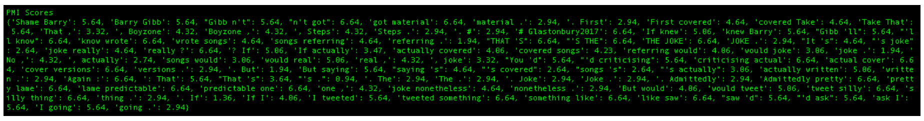
Below is the wordcloud command I used in R; the Python code used to generate the input text (and all other code for Q1) based on the TF-IDF Matrix values can be found in q1.py.

```
wordcloud("Shame Barry Barry Barry Gibb Gibb Gibb n't got material First covered
     covered Take That That That , , , , , , , Boyzone Steps # Glastonbury2017 If If If
     If knew 'll know wrote songs songs referring referring referring THAT 'S THE JOKE
     It 's 's 's 's 's joke joke joke joke joke really ? actually actually actually
     actually would would would would No real You 'd 'd 'd criticising actual cover
     versions But But But saying written Again : The Joke Admittedly pretty lame
     predictable one nonetheless tweet silly thing I I I I I tweeted something like
     saw ask going", colors=brewer.pal(6, "Dark2"),random.order=FALSE, rot.per=0,
     scale=c(4, .0625), min.freq=1)
```

I think the TF-IDF method improves on the TF method, however, I would probably conduct this with ngrams instead of single words to avoid the highlighting of words used for emphasis e.g. 'that' as having more importance than 'song'. Of course, taken in a broader context, that certain terms used for emphasis show up as significant under both methods is rather telling of both the tone and intention of the text exchange; my choice of how to conduct the analysis, as always, would largely hinge on what I was hoping to achieve. But it seems fair to say, as is suggested by the development of the technique, that TF-IDF provides more and better information than TF alone.

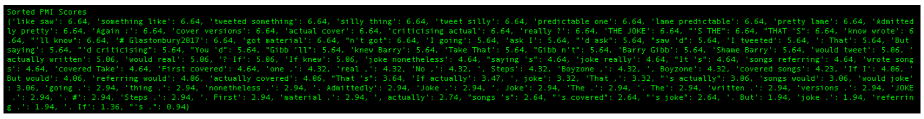
2: PMI scores

The code for Q2 can be found in q2.py. There is too much to include here, but it includes methods to generate the PMI scores, as well as an implementation of merge sort for Python dictionaries that makes the presentation of the top 10 direct and simple.



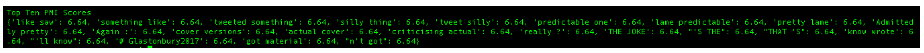
```
PMI Scores
{'Shame Barry': 5.64, 'Barry Gibb': 5.64, "Gibb n't": 5.64, "n't got": 6.64, 'got material': 6.64, 'material .': 2.94, '. First': 2.94, 'First covered': 4.64, 'covered Take': 4.64, 'Take That':
 5.64, 'That ,': 3.32, ', Boyzone': 4.32, 'Boyzone ,': 4.32, ', Steps': 4.32, 'Steps .': 2.94, '. #': 2.94, '# Glastonbury2017': 6.64, 'If knew': 5.06, 'knew Barry': 5.64, "Gibb 'll": 5.64, "'l
l know": 6.64, 'know wrote': 6.64, 'wrote songs': 4.64, 'songs referring': 4.64, 'referring .': 1.94, "THAT 'S": 6.64, "'S THE": 6.64, 'THE JOKE': 6.64, 'JOKE .': 2.94, "It 's": 4.64, "'s joke"
: 2.64, 'joke really': 4.64, 'really ?': 6.64, '? If': 5.06, 'If actually': 3.47, 'actually covered': 4.06, 'covered songs': 4.23, 'referring would': 4.06, 'would joke': 3.06, 'joke .': 1.94, '
No ,': 4.32, ', actually': 2.74, 'songs would': 3.06, 'would real': 5.06, 'real ,': 4.32, ', joke': 3.32, "You 'd": 5.64, "'d criticising": 5.64, 'criticising actual': 6.64, 'actual cover': 6.6
4, 'cover versions': 6.64, 'versions .': 2.94, '. But': 1.94, 'But saying': 5.64, "saying 's": 4.64, "'s covered": 2.64, "songs 's": 2.64, "'s actually": 3.06, 'actually written': 5.06, 'writte
n .': 2.94, 'Again :': 6.64, ': That': 5.64, "That 's": 3.64, "'s .": 0.94, '. The': 2.94, 'The .': 2.94, '. Joke': 2.94, 'Joke .': 2.94, '. Admittedly': 2.94, 'Admittedly pretty': 6.64, 'prett
y lame': 6.64, 'lame predictable': 6.64, 'predictable one': 6.64, 'one ,': 4.32, 'joke nonetheless': 4.64, 'nonetheless .': 2.94, 'But would': 4.06, 'would tweet': 5.06, 'tweet silly': 6.64, 's
illy thing': 6.64, 'thing .': 2.94, '. If': 1.36, 'If I': 4.06, 'I tweeted': 5.64, 'tweeted something': 6.64, 'something like': 6.64, 'like saw': 6.64, "saw 'd": 5.64, "'d ask": 5.64, 'ask I':
5.64, 'I going': 5.64, 'going .': 2.94}
```

Figure 3: PMI Scores



```
Sorted PMI Scores
{'like saw': 6.64, 'something like': 6.64, 'tweeted something': 6.64, 'silly thing': 6.64, 'tweet silly': 6.64, 'predictable one': 6.64, 'lame predictable': 6.64, 'pretty lame': 6.64, 'Admitted
ly pretty': 6.64, 'Again :': 6.64, 'cover versions': 6.64, 'actual cover': 6.64, 'criticising actual': 6.64, 'really ?': 6.64, 'THE JOKE': 6.64, "'S THE": 6.64, "THAT 'S": 6.64, 'know wrote': 6
.64, "'ll know": 6.64, '# Glastonbury2017': 6.64, 'got material': 6.64, "n't got": 6.64, 'I going': 5.64, 'ask I': 5.64, "'d ask": 5.64, "saw 'd": 5.64, 'I tweeted': 5.64, ': That': 5.64, 'But
saying': 5.64, "'d criticising": 5.64, "You 'd": 5.64, "Gibb 'll": 5.64, 'knew Barry': 5.64, 'Take That': 5.64, "Gibb n't": 5.64, 'Barry Gibb': 5.64, 'Shame Barry': 5.64, 'would tweet': 5.06, '
actually written': 5.06, 'would real': 5.06, '? If': 5.06, 'If knew': 5.06, 'joke nonetheless': 4.64, "saying 's": 4.64, 'joke really': 4.64, "It 's": 4.64, 'songs referring': 4.64, 'wrote song
s': 4.64, 'covered Take': 4.64, 'First covered': 4.64, 'one ,': 4.32, 'real ,': 4.32, 'No ,': 4.32, ', Steps': 4.32, 'Boyzone ,': 4.32, ', Boyzone': 4.32, 'covered songs': 4.23, 'If I': 4.06, '
But would': 4.06, 'referring would': 4.06, 'actually covered': 4.06, "That 's": 3.64, 'If actually': 3.47, ', joke': 3.32, 'That ,': 3.32, "'s actually": 3.06, 'songs would': 3.06, 'would joke'
: 3.06, 'going .': 2.94, 'thing .': 2.94, 'nonetheless .': 2.94, '. Admittedly': 2.94, 'Joke .': 2.94, '. Joke': 2.94, 'The .': 2.94, '. The': 2.94, 'written .': 2.94, 'versions .': 2.94, 'JOKE
 .': 2.94, '. #': 2.94, 'Steps .': 2.94, '. First': 2.94, 'material .': 2.94, ', actually': 2.74, "songs 's": 2.64, "'s covered": 2.64, "'s joke": 2.64, '. But': 1.94, 'joke .': 1.94, 'referrin
g .': 1.94, '. If': 1.36, "'s .": 0.94}
```

Figure 4: PMI Scores Sorted



```
Top Ten PMI Scores
{'like saw': 6.64, 'something like': 6.64, 'tweeted something': 6.64, 'silly thing': 6.64, 'tweet silly': 6.64, 'predictable one': 6.64, 'lame predictable': 6.64, 'pretty lame': 6.64, 'Admitted
ly pretty': 6.64, 'Again :': 6.64, 'cover versions': 6.64, 'actual cover': 6.64, 'criticising actual': 6.64, 'really ?': 6.64, 'THE JOKE': 6.64, "'S THE": 6.64, "THAT 'S": 6.64, 'know wrote': 6
.64, "'ll know": 6.64, '# Glastonbury2017': 6.64, 'got material': 6.64, "n't got": 6.64}
```

Figure 5: PMI Scores Top Ten

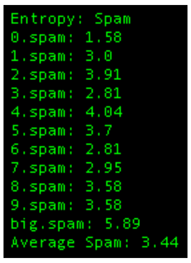
The results do make sense (at least as much as one can hope from a hilarious and grumpy twitter exchange). Using a minimal cut-off frequency does change the results, but given that most terms are used only once, the resulting information is rather divorced from the original text and of little use. More to the point, occurring once is not a low-frequency in this corpus. What's encouraging about the PMI results is that the top ten terms also represent phrases (or sub-phrases) crucial to the original poster trying desperately to get idiots to recognise that she made a great joke and they don't get it (also, you see similar elements from the dullards hoping to make their case). It would seem that in general doing ngram analysis with PMI scores could be quite useful in delineating high-impact phrases.

3: Entropy

Source materials are documented in random.txt and spam.txt. All code used for this section is found in q3.py, but below is the key element of the entropy calculation:

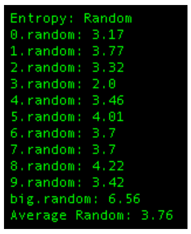
```
def entropy(file_name):
  tokens = tokenize_text_file_remove_stop_words(file_name)
  frequencies = FreqDist(tokens)
  probabilities = [frequencies.freq(p) for p in frequencies]
  return round(-sum(p * math.log(p, 2) for p in probabilities),2)
```

This could have been done manually, but I used the FreqDist class within nltk to become better acquainted with the features of nltk; FreqDist allowed me to quickly and cleanly generate the probabilities of each token. Beyond that, I used basic features of Python to automate the calculation of entropy for the spam-set, random-set, and combined-set.



```
Entropy: Spam
0.spam: 1.58
1.spam: 3.0
2.spam: 3.91
3.spam: 2.81
4.spam: 4.04
5.spam: 3.7
6.spam: 2.81
7.spam: 2.95
8.spam: 3.58
9.spam: 3.58
big.spam: 5.89
Average Spam: 3.44
```

(a) Entropy: Space



```
Entropy: Random
0.random: 3.17
1.random: 3.77
2.random: 3.32
3.random: 2.0
4.random: 3.46
5.random: 4.01
6.random: 3.7
7.random: 3.7
8.random: 4.22
9.random: 3.42
big.random: 6.56
Average Random: 3.76
```

(b) Entropy: Random

Figure 6: Entropy Outputs

The entropy of the combined set is 7.09, while the averages for spam and random are 3.44 and 3.76, respectively. While I wouldn't interpret the combined set as the most interesting, this score reflects my expectation that it does have the most diversity: it's combining random tweets with spam tweets for a particular product.

What's interesting is how close the spam and random sets are to each other on average. You'll notice that the combined spam and combined random are 5.89 and 6.56, respectively. I'd expect most combined Tweets to be more diverse than any of the individual components, but I was surprised that a spam set for the same product was close to being as diverse as the random sample.

On reflection, I think few things are at play, including:

- uniformity of random
  - my "random" tweets reflect those Twitter expects will resonate with me; it's not unsurprising they will have a uniformity similar to a targeted advertisement
  - similarly, and perhaps more insidiously, the advertisements might be designed to mimic, in some way, the content I choose to review on Twitter
- generalised nature of the spam business
  - the firm trying to sell to me has a fairy malleable product, so advertising it could naturally lead to diverse statements
  - but see above for a more likely explanation
- advertisers getting better i.e. making ad-Tweets seem like normal Tweets
  - this point was already made above, but is worth mentioning again: without it, Twitter and advertisers will struggle at their business