Practical 4

Question 1:

1a.) Here we have to get the data. I opted to use real tweets about the most recent presidential inauguration. These were in a csv format and retrieved from Kaggle. The CSV data was extracted through python (figure 1). After data extraction, the next step was to put the tweet data in a txt file which could then be manipulated in python and R. However, it must be cleaned first. To do this, each text item must be investigated. Luckily, all tweets in question began with RT so splitting them into different text items on the RT was simple. It would have been a different code if there hadn't been such an easy text-item division marker. The stopwords used were from the nltk.corpus library. Useless characters were also marked. Twitter handles, marked by the '@' are removed later in this code because they don't carry much meaning, they are merely names, and thus mean nothing other than to identify the person in question. Here, the shared topic of the tweets is not named based (e.g. all tweets @realdonaldtrump) so handles were deleted (process seen figure 2).

```
In [11]: with open('inaugurationA.csv', 'rb') as f:
             result = chardet.detect(f.read())
         #take the first ten examples
         df = pd.read_csv('inaugurationA.csv', encoding=result['encoding'], nrows=10)

In [12]: text_data = df.iloc[:,1]
         pd.DataFrame(text_data)

Out[12]:
```

	text
0	RT @9gagtv: President Donald Trump Accidentall...
1	RT @SpcStevens: This makes it much easier. #In...
2	RT @Claire_Phipps @nwg83 so what youre saying ...
3	RT @JahovasWitniss: Wait... what!?\r #Inaugura...
4	RT @3lectric5heep: BREAKING: Here's The Full L...
5	RT @DanJGross: Crowd count is in:\r Trump 2017...
6	RT @FunnyVines: The 45th President of The Unit...
7	RT @JCRaskaus: BREAKING: Here's The Full List ...
8	RT @LessGovMoreFun: BREAKING: Here's The Full ...
9	RT @WBLZSportsChica: 1 million #Patriots fans ...

Figure 1: Tweets from the Kaggle CSV in Python

```
List = open("practical_4_q1.txt").read()
List = List.split('RT')
```

```
#get rid of URLs
new_list=[]
for sentence in List:
    URLless_string = re.sub(r'\w+:\/{2}[\d\w-]+(\.[\d\w-]+)*(?:(?:\/[^\s/]*))*', '', sentence)
    sentence = URLless_string
    new_list.append(sentence)
```

```
#get rid of stopwords and useless characters
useless_char =['\n','[',']','\'','\"',':','(',')',':','#']
stop = set(stopwords.words('english'))
txt = []
for sent in new_list:
    for ch in useless_char:
        sent=sent.replace(ch," ")
    no_stops = [i for i in sent.lower().split() if i not in stop and i != 'rt'] #term rt unecessary. Might change results.
    txt.append(no_stops)
```

```
#get rid of twitter handles because they aren't real words and are only useful to twitter - might change results
cleaned_tweets = []
for sentence in txt:
    new_list = [word for word in sentence if not word.startswith('@')]
    j = " ";
    no_handles = j.join(new_list)
    cleaned_tweets.append(no_handles)
```

Figure 2: Code for cleaning the Data

1b.) Using the txt file derived from the above code, I made a document term matrix and frequency table in R (figure 3). The list of words in the corpus and their respective frequencies are in figure 4.

```
> tweet_dtm <- DocumentTermMatrix(tweets)
> tweet_dtm
<<DocumentTermMatrix (documents: 10, terms: 38)>>
Non-/sparse entries: 61/319
Sparsity           : 84%
Maximal term length: 12
Weighting          : term frequency (tf)
> freq <- colSums(as.matrix(tweet_dtm))
> ord <- order(freq)
> length(freq)
[1] 38
> table(freq)
freq
 1  2  3  4 10
28  4  4  1  1
>
```

Figure 3: The Document Term Matrix of the Corpus

```
> freq
accidentally         bane       donald inauguration    president       quotes       speech
           1            1            1           10            2            1            1
       trump         fans          lol      million       parade     patriots        still
           2            1            1            1            1            1            1
      easier        makes         much     attended       people       period       saying
           1            1            1            1            4            1            1
     watched        youre         wait         what     arrested     breaking         full
           1            1            1            1            3            3            3
        list        riots        count        crowd        obama      america       states
           3            2            1            1            2            1            1
      united      gotnews       djtâ€
           1            1            1
> |
```

Figure 4: List of the words in the Corpus and their frequencies

Additionally, I made an R wordcloud with a min.freq = 1. Initially the min.freq was equal to 2, but this made for quite the sparsely populated cloud, as can be seen from the above dtm. Although by and large this worked, one error regarding the most common word (inauguration) appeared:

```
> wordcloud(names(freq), freq, min.freq=1, scale=c(5, .1), colors= palette())
Warning message:
In wordcloud(names(freq), freq, min.freq = 1, scale = c(5, 0.1),  :
  inauguration could not be fit on page. It will not be plotted.
> |
```

A stackoverflow search revealed that usually this error occurs when the scale is too large, and advised using a smaller scale. However, using the scale provided in the proposed solution (scale=c (4, .5)), as well as the exact code provided in the lecture 3 slides, yielded the same results. Investigation into the txt file as a corpus revealed nothing out of the ordinary. Copying and pasting the context of the txt file into the wordcloud command had the same results. I am curious as to why this is. The wordcloud achieved is seen in figure 5. Finally, a complete term frequency matrix was created (figure 6).

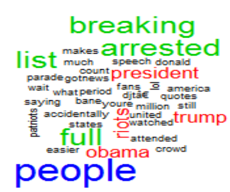

Fig.5: Wordcloud (sans 'Inauguration')

```
> (tf <- as.matrix(tweet_dtm))
                      Terms
Docs                   accidentally bane donald inauguration president quotes speech trump fans lol million parade patriots still easier
  cleanedTweets-1.txt             1    1      1            1         1      1      1     1    0   0       0      0        0     0      0
  cleanedTweets-10.txt            0    0      0            1         0      0      0     0    1   1       1      1        1     1      0
  cleanedTweets-2.txt             0    0      0            1         0      0      0     0    0   0       0      0        0     0      1
  cleanedTweets-3.txt             0    0      0            1         0      0      0     0    0   0       0      0        0     0      0
  cleanedTweets-4.txt             0    0      0            1         0      0      0     0    0   0       0      0        0     0      0
  cleanedTweets-5.txt             0    0      0            1         0      0      0     0    0   0       0      0        0     0      0
  cleanedTweets-6.txt             0    0      0            1         0      0      0     1    0   0       0      0        0     0      0
  cleanedTweets-7.txt             0    0      0            1         1      0      0     0    0   0       0      0        0     0      0
  cleanedTweets-8.txt             0    0      0            1         0      0      0     0    0   0       0      0        0     0      0
  cleanedTweets-9.txt             0    0      0            1         0      0      0     0    0   0       0      0        0     0      0
                      Terms
Docs                   makes much attended people period saying watched youre wait what arrested breaking full list riots count crowd
  cleanedTweets-1.txt      0    0        0      0      0      0       0     0    0    0        0        0    0    0     0     0     0
  cleanedTweets-10.txt     0    0        0      0      0      0       0     0    0    0        0        0    0    0     0     0     0
  cleanedTweets-2.txt      1    1        0      0      0      0       0     0    0    0        0        0    0    0     0     0     0
  cleanedTweets-3.txt      0    0        1      1      1      1       1     1    0    0        0        0    0    0     0     0     0
  cleanedTweets-4.txt      0    0        0      0      0      0       0     0    1    1        0        0    0    0     0     0     0
  cleanedTweets-5.txt      0    0        0      1      0      0       0     0    0    0        1        1    1    1     1     0     0
  cleanedTweets-6.txt      0    0        0      0      0      0       0     0    0    0        0        0    0    0     0     1     1
  cleanedTweets-7.txt      0    0        0      0      0      0       0     0    0    0        0        0    0    0     0     0     0
  cleanedTweets-8.txt      0    0        0      1      0      0       0     0    0    0        1        1    1    1     1     0     0
  cleanedTweets-9.txt      0    0        0      1      0      0       0     0    0    0        1        1    1    1     0     0     0
                      Terms
Docs                   obama america states united gotnews djtâ€
  cleanedTweets-1.txt      0       0      0      0       0     0
  cleanedTweets-10.txt     0       0      0      0       0     0
  cleanedTweets-2.txt      0       0      0      0       0     0
  cleanedTweets-3.txt      0       0      0      0       0     0
  cleanedTweets-4.txt      0       0      0      0       0     0
  cleanedTweets-5.txt      0       0      0      0       0     0
  cleanedTweets-6.txt      2       0      0      0       0     0
  cleanedTweets-7.txt      0       1      1      1       0     0
  cleanedTweets-8.txt      0       0      0      0       1     0
  cleanedTweets-9.txt      0       0      0      0       0     1
```

Fig. 6: Term Frequency Matrix for the Tweets Investigated

1c.) Using the above TF Matrix, and code adapted for my particular matrix construction from Liu (November 17, 2015), the idf weights for all terms were computed. But to get the tf-idf, I used the SnowballC R library as per advice from Amazon WebServices (n.d.). I assigned tweet_dtm_tfidf to DocumentTermMatrix(tweets, control=list(weighting=weightTfIdf)). This made a document term matrix like the one in part 1b, but instead of just using all terms, I added the idf weights which had been previously found. Although I had calculated the idf weights previously, this was unnecessary as the weightTfIdf did this for me. Inspecting tweet_dtm_tfidf reveals the tf-idf scores (figure 7).

```
> inspect(tweet_dtm_tfidf)
<<DocumentTermMatrix (documents: 10, terms: 38)>>
Non-/sparse entries: 51/329
Sparsity           : 87%
Maximal term length: 12
Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
Sample             :
                     Terms
Docs                   arrested  breaking   easier    makes     much    obama    people president     wait     what
  cleanedTweets-1.txt  0.0000000 0.0000000 0.000000 0.000000 0.000000 0.000000 0.0000000 0.2902410 0.000000 0.000000
  cleanedTweets-10.txt 0.0000000 0.0000000 0.000000 0.000000 0.000000 0.000000 0.0000000 0.0000000 0.000000 0.000000
  cleanedTweets-2.txt  0.0000000 0.0000000 0.830482 0.830482 0.830482 0.000000 0.0000000 0.0000000 0.000000 0.000000
  cleanedTweets-3.txt  0.0000000 0.0000000 0.000000 0.000000 0.000000 0.000000 0.1888469 0.0000000 0.000000 0.000000
  cleanedTweets-4.txt  0.0000000 0.0000000 0.000000 0.000000 0.000000 0.000000 0.0000000 0.0000000 1.107309 1.107309
  cleanedTweets-5.txt  0.2481379 0.2481379 0.000000 0.000000 0.000000 0.000000 0.1888469 0.0000000 0.000000 0.000000
  cleanedTweets-6.txt  0.0000000 0.0000000 0.000000 0.000000 0.000000 1.107309 0.0000000 0.0000000 0.000000 0.000000
  cleanedTweets-7.txt  0.0000000 0.0000000 0.000000 0.000000 0.000000 0.000000 0.0000000 0.4643856 0.000000 0.000000
  cleanedTweets-8.txt  0.2171207 0.2171207 0.000000 0.000000 0.000000 0.000000 0.1652410 0.0000000 0.000000 0.000000
  cleanedTweets-9.txt  0.2481379 0.2481379 0.000000 0.000000 0.000000 0.000000 0.1888469 0.0000000 0.000000 0.000000
> |
```

Fig. 7: Matrix of tf-idf scores

There are 38 terms, but the program only displays the tf-idfs of 10 of them. The resulting word-cloud in light of these tf-idf scores is seen in figure 8. The words in this word cloud are of 10 terms in the corpus. They are different in that they are not all the most commonly used, but they are commonly used within a single or only a few tweets, which means that they carry meaning. This is the phenomena tf-idf attempts to capture and convey.

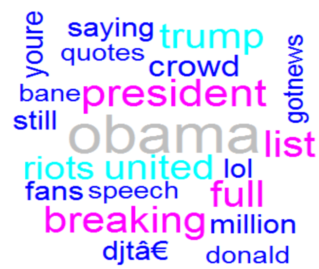

Fig. 8: wordcloud of tf-idf scores

Question 2:

Pointwise mutual information measures association between words. We pick two adjacent words in a corpus (bigrams), take the count of word 1, the count of word 2, and the count of co-occurrences to calculate the PMI. Luckily, module nltk.collocations does this for me. The code (figure 9) is in python and inspired by code given by Rob Neuhaus on stackoverflow (Dec 30, 2011). Print(prefix_keys) yielded a list of all bigrams and their association scores sorted from strongest to weakest (top 10 in figure 10). As a side-note, a new line of code had to be added to the original data fetching code shown in part 1a to remove numbers from the corpus. This had been done via R but not yet in python and it was interesting to try both ways. Although a look at the corpus will tell you that these combinations do make sense (bigrams breaking-full, full-list, and list-people appear twice in the corpus and the others once), the fact that the most commonly occurring bigram only appear twice means that it is not a particularly good corpus to use as an example to test this concept. Sources suggest that using a bigger corpus would result in more 'accurate' results as PMI is biased towards infrequent events (Turney, P. D., & Pantel, P, 2010). Changing the minimal cut-off frequency will not solve the problem – variety and more examples can only occur when more examples are introduced to the corpus.

```python
import nltk.collocations
import nltk.corpus
import collections
```

```python
f = open('new_corpus.txt').read()
f = f.split()
```

```python
bgm = nltk.collocations.BigramAssocMeasures()
find = nltk.collocations.BigramCollocationFinder.from_word
scored = find.score_ngrams( bgm.likelihood_ratio  )
```

```python
# Group bigrams by 1st word in bigram.
prefix_keys = collections.defaultdict(list)
for key, scores in scored:
    prefix_keys[key[0]].append((key[1], scores))
```

```python
# Sort keyed bigrams by strongest association.
for key in prefix_keys:
   prefix_keys[key].sort(key = lambda x: -x[1])
print(prefix_keys)
```

Fig.9: Python Code to find PMI scores

```
'breaking': [('full', 24.121954326672384)]
'full': [('list', 24.121954326672384)]
'list': [('people', 19.623273169721916)]
'people': [('arrested', 19.623273169721916)]
'riots': [('dc', 17.73577537394937)]
'arrested': [('inauguration', 11.904668285574514)]
'accidentally': [('quotes', 10.270311779741764)]
'crowd': [('count', 10.270311779741764)]
'fans': [('parade', 10.270311779741764)]
'million':[('patriots', 10.270311779741764)]
```

Fig.10: list of top ten PMI scores in corpus

Question 3:

Using user marmotter's code (figure 11) April, 2016 on r/dailyprogrammer, I calculated the entropy for the following lists of tweets:

Spam

OMG skinny coffee made me lose seven pounds in seven days! skinny coffee revolution!

Thank GOD skinny coffee ships worldwide! Don't know what I'd do without my skinny coffee!!

steveieee Get skinny coffee challenge and Burn Excess fat without skipping your fav meals! twenty dollars!

I lost seven pounds of fat in seven days! unreal!! skinny coffee is a miracle!

Get skinny coffee challenge everywhere! ship free worldwide!

seven day challenge, lose those seven pounds, girl!

Aury9forever Get skinny coffee challenge and Burn Excess fat without skipping your fav meals! twenty dollars!

xxellttil Get skinny coffee challenge and Burn Excess fat without skipping your fav meals!twenty dollars!

Never skip your favourite meals just to lose weight,take a challenge on us.

struggling with being over weight? Lose seven pounds in seven days for twenty dollars! Free shipping worldwide!

Not Spam

Feeling really blue today :(can't wait til my bae gets home!

TFW you wake up and the day already seems overwhelming

There is seriously nothing like coffee, the sunrise, and a good book

Some of y'all be so shallow! Don't hate me cause you ain't me!

oh my god, my boy Lamar be killin it!

If you live to be 100 you should just make up some fake reason why just to mess with peoples heads

Been on hold so long I can't remember who I even called smh

waiter, uh, theres a reflection of a sad lonely man in my soup?

Relationships are mostly you apologizing for saying something hilarious

What do you mean I didn't win I ate more wet t-shirts than anyone else

```python
from __future__ import division
import math
```

```python
def inputLength(spam):
    return float(len(str(spam)))

def dictCreate(spam):
    freq_dict = {}
    for key in str(spam):
        freq_dict.setdefault(key, 0)
        freq_dict[key] = freq_dict[key] + (1 / inputLength(spam))
    return freq_dict.values()

def main(spam):
    entropy = 0.0
    for v in dictCreate(spam):
        entropy = entropy + (v * math.log2(v))
    return entropy * -1
```

Fig.11: Code to find entropy of a string

The results were spam with an entropy score of 4.2801307214326645 and not spam (ham!) with a score of 4.049611539276234 and a combined entropy of 4.2085382753968075. This doesn't make much sense to me because I was under the impression that the less alike or the greater the change in a list of strings, the greater the entropy, and inversely, the more similar the words, the less the entropy. As you can see by my text samples, the spam tweets are very similar, nigh identical, whereas the ham tweets are not. The function calculating entropy is correct because it is using the literal math of entropy and it passes a test that I designed so I am unsure as to what is going on. At first, I considered that it could possibly be the way I was reading in the txt file, and indeed at first it was problematic because I was reading in the full tweet as an item, not the individual words. But once that was ameliorated, I'm not sure what is wrong. Should I have more time at a later date, I would love to play around more with this. I will definitely be fielding this question at the next practical.

.

Works Cited

Amazon Web Services. (n.d.). Basic Text Mining with R. Retrieved October 12, 2017, from https://rstudio-pubs-static.s3.amazonaws.com/132792_864e3813b0ec47cb95c7e1e2e2ad83e7.html

Liu, E. (2015, November 17). TF-IDF, Term Frequency-Inverse Document Frequency. Retrieved October 12, 2017, from http://ethen8181.github.io/machine-learning/clustering_old/tf_idf/tf_idf.html

Marmotter. (2016, April 18). Challenge #263 [Easy] Calculating Shannon Entropy of a String • r/dailyprogrammer. Retrieved October 12, 2017, from https://www.reddit.com/r/dailyprogrammer/comments/4fc896/20160418_challenge_263_easy_calculating_shannon/

Turney, P. D., & Pantel, P. (2010). From frequency to meaning: Vector space models of semantics. Journal of artificial intelligence research, 37, 141-188.