Q1. **Find 10 short text-items (20-30 words); they could be emails, short docs,tweets or whatever… Make sure they all deal with some common topic of interest; so they have some of the same words.**

1. **Remove the standard stopwords from them using some standard list, use nltk.**

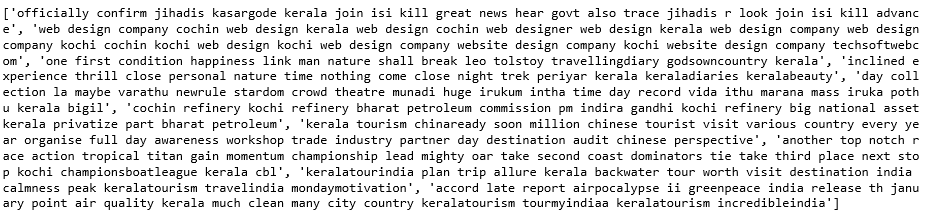
**Ans1a.**

10 short tweets, all similar to the topic ‘Kerala’ are taken in a dictionary. Pre - processing of the input from the dictionary is done which comprises of:

**Tokenization followed by normalization (down case-> remove digits and special characters -> stopwords removal ) followed by pos-tagging and Lemmatization through WordNet Lemmatizer using the nltk package functions.**

A list of lists is generated having all the pre- processed tweets. Then all the documents are collaborated to one final corpus.

A snippet of final pre-processed output is shown below:



1. **Compute the TF scores for all the remaining words in the texts and use R to show the word-cloud for these words. In your answer provide the matrix of TF scores and the word-cloud image.**

**Ans1b.**

**TF Score:** Term Frequency refers to the count of a term (word) in a given document (text-item).

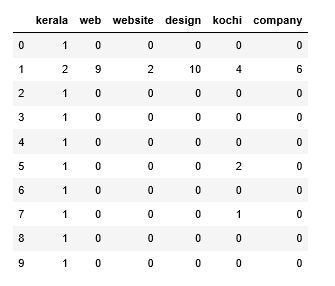
It is calculated by: Term frequency, tf (Term t, in Doc d) = Frequency, f(Term t,in doc d).

In order to calculate the TF Score using the nltk package, CountVectorizer module from sklearn.feature\_extraction.text is used.

The fit\_transform () helps to calculate the TF Score of the final pre - processed output string.

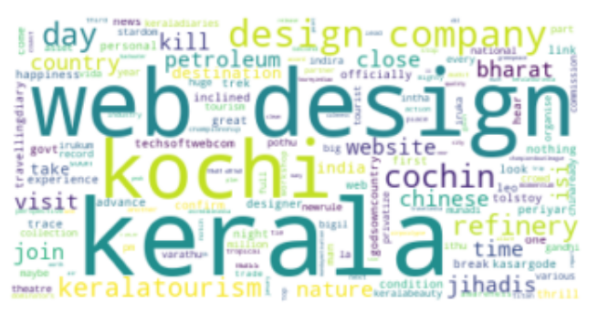
A short snippet of TF Scores from the large dataset containing the TF Scores of all the remaining words is shown below:

The matrix depicts left side with all 10 small documents(0-9) and on the top, there exists words(prominent in the wordcloud) and their frequencies in each document. Say the term ‘kerala’ occurs twice in second document and once in every other document.



Wordcloud depicts the visual representation of the words of all the remaining words. Here it displays that few terms like kerala, kochi, web,design are most frequent among the documents.

A wordcloud of all the remaining words is shown below:

c. **Now, compute the TF-IDF scores for all the same words in the texts. Construct a set of words that represents the TF-IDF scores you have found, for all the words. Use R to show a word-cloud for these words. Also, provide the matrix of TF-IDF scores and the word-cloud image.**

**Ans1c.**

**IDF:** The inverse document frequency is a measure of how much information the word provides, i.e., if it's common or rare across all documents. It uses the size of the document set, and the frequency of the term’s occurrence in set of documents (text-items). IDF bit is used to weight the frequency by rarity i.e. A term that is really frequent in a given document, but really rare in every other document is not-equal-to a term that is really frequent in every document. A term that is in every text-item has no weight .

It is calculated by : log(Total number of docs in the corpus N/Number of docs where term t appears).

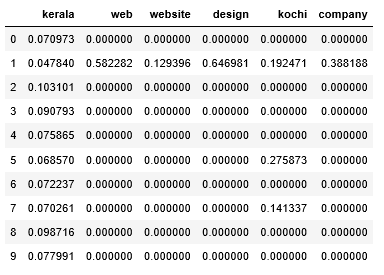
TF-IDF is calculated by product of term frequency and the Inverse Document frequency.

Also it is calculated using the TfidfTransformer module of sklearn.feature\_extraction.text package.

The fit\_transform () of the TfidfTransformer helps to calculate the TF Score of the final pre - processed output string and stored in a data storage data frame.

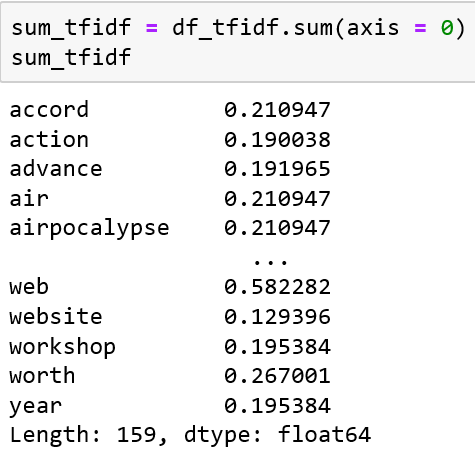
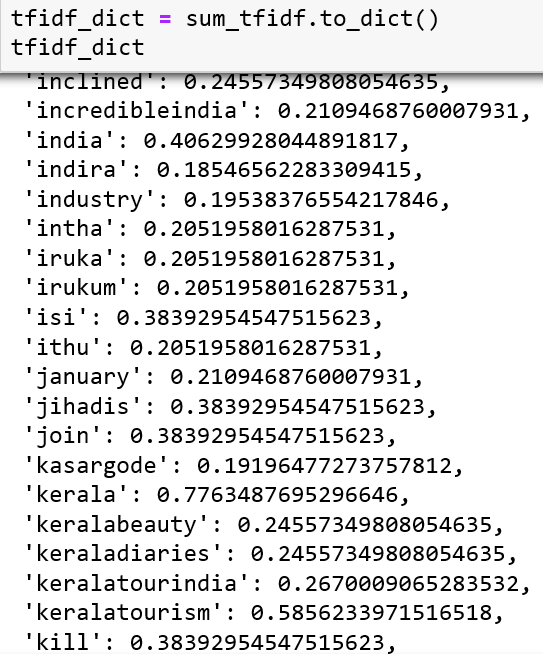
A snippet of TF-IDF is shown below:

The matrix depicts left side with all 10 small documents(0-9) and on the top, there exists words(prominent in the wordcloud) and their TF IDF values across all documents. How important a word is to a document in a collection or corpus is shown.



Summation of tf-idf for all the words is done and is converted to a dictionary form so that it can be fed to wordcloud while generating using the frequencies.

Snippet is shown below:

In the end, a wordcloud is generated using the all the words.



==================================================================================

Q2. **Using Python or R, compute the PMI scores for all adjacent pairs of words in your 10-doc corpus (ie the texts after stop-word removal).**

**List the top-10 pairs based on the PMI scores found for the pairs.**

**Do the results make sense? If not, then introduce a minimal cut-off frequency and re-compute the top-10 until they seem sensible.**

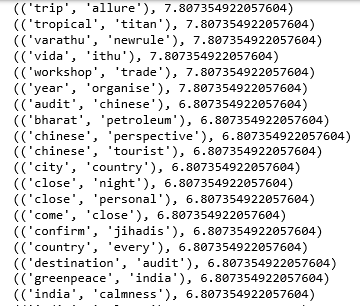
**Ans2.** Pointwise Mutual Information, PMI is defined as the measure of association. It measures how much more likely the words co-occur than if they were independent in the text.

It is calculated as: PMI(word1, word2) = Probability(word1, word2)/ Prob(word1) P(Word2).

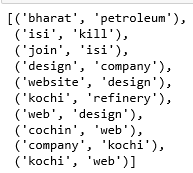
PMI is calculated by nltk.collocations.BigramAssocMeasures() module

Corpus with 10 documents after pre- processing are converted to a string and tokenised which is fed to BigramCollocationFinder. First of all, Bigrams are formed using BigramCollocationFinder. Further pmi score is calculated using the bigram\_measures.pmi.

A snippet of bigrams along with their PMIs is shown below:



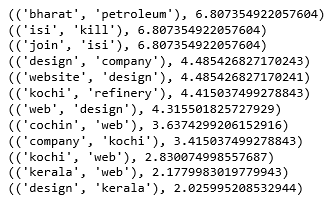
Further, top 10 pairs are fetched from the output which are having the same PMI.



The issue with PMI is that it over-estimates the low frequency events because of how it treats counts. So the result doesn’t make any sense. The solution to this issue is to have a minimum frequency cut off.

So to achieve that apply\_freq\_filter() is used by which all the collocations that had a frequency of less than 2 don't exist in this list.

Snippet of new 10 pairs is shown below:



==================================================================================

**Q3.Entropy has been used to determine whether tweet set is interesting (contains variety) or repetitive (spam). Create two sets of 10 made-up tweets:**

**a. spam-set: where the 10 tweets are very similar containing an advert for a product.**

**b. random-set: where the 10 tweets are very different, chosen at random from Twitter.**

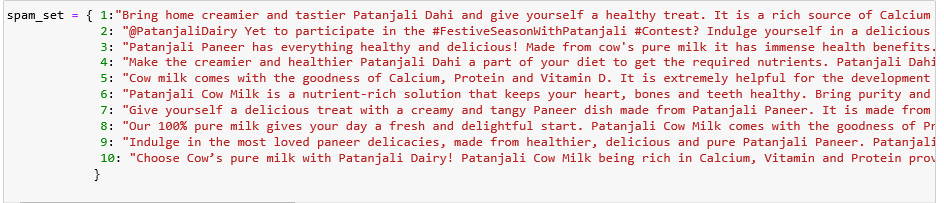
**Now, find a Python/R program or package that computes entropy and find the entropy values for (i) spam-set, (ii) random-set, (iii) the two sets combined.**

**Report the program you used and its source, the tweet data and the entropy values found.**

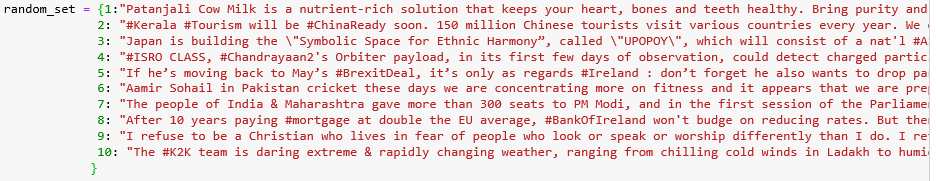
**Ans3.**

Two sets (spam and random) are taken in consideration.

Spam set consists of 10 made up tweets which are similar to each other and containing the advert for a product. Further the spam set is undergone pre-processing(tokenised-> normalized ( downcase, digits and special characters removal -> stopwords removal) and finally joined to form a single string.)

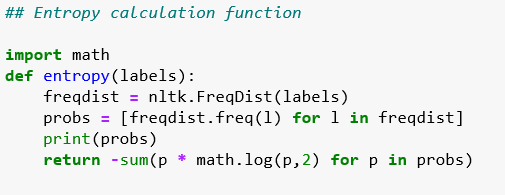


Random set consists of 10 made up tweets which are dissimilar to each other and completely random. Further the random set is undergone pre-processing (tokenised-> normalized ( downcase, digits and special characters removal -> stopwords removal) and finally joined to form a single string.)



Entropy is defined as the sum of the probability of each label times the log probability of that same label, written as H(A) = -sum(p \* log(p), axis=0). It is a measure of randomness of all the words.

Program that computes the entropy is shown below in which a list of tokenised words is passed to the entropy function(). FreqDist calculates the frequency distribution for each token and prob generates the probability of each token and entropy is returned(-sum(p\*log(p with base 2))) for every probability. [https://www.nltk.org/book/ch06.html]



1. Entropy is calculated for the spam set is shown below:

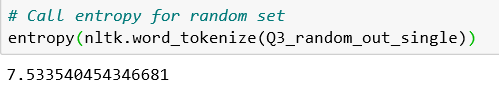
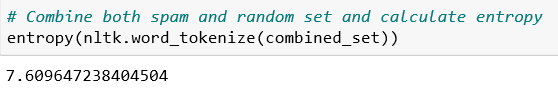
Here the entropy is 5.97 which indicates the less randomness among the data because the data here is similar to each other.

1. Entropy is calculated for the random set is shown below:

Here the entropy is 7.533 which indicates the good degree of randomness among the data because the data here is different from each other.

1. Entropy is calculated for the combination of spam and random set is shown below:



==================================================================================

**REFERENCES**

1. (Natural Language Processing with Python,2019 , [Steven Bird](http://stevenbird.net/), [Ewan Klein](http://homepages.inf.ed.ac.uk/ewan/) and [Edward Loper](http://ed.loper.org/))