NATURAL LANGUAGE PROCESSING – CSE4022

FALL 2020-2021

SLOT – E2+TE2

TEAM – 3

TEAM 3			
21	BCE	18BCE0408	HRISHITA CHAKRABARTI
22	BCE	18BCE0458	NAVDEEP CHAWLA
23	BCE	18BCE0486	ANIKET KUMAR
24	BCE	18BCE0490	SHIVAM ANAND
25	BCE	18BCE0492	ABHISHEK KUSHWAHA
26	BCE	18BCE0512	SAGAR PRATEEK
27	BCE	18BCE0536	NAYNIKA WASON
28	BCE	18BCE0548	SHOUNAK SAURAV BHATTACHARYA
29	BCE	18BCE0589	KASAMSETTY SAI PREETHAM
30	BCE	18BCE0609	RITESH CHOWDARY KONA KANCHI

PROBLEM STATEMENT:

Assume you are a part of the NLP Tech team that works for a Publishing House. There is a shortlisted applicant (with her writing samples) for the Editor-in-chief position. How can you help the publishing house with the decision on hiring this applicant?

SOLUTION:

https://colab.research.google.com/drive/10UERBcTyTfw0I4Z0x7JpAaDB45UYwf7J?usp=sharing

1. DATA ACQUISITION

Dataset of text files (or say writing samples of the shortlisted applicant) has been extracted from links given below.

- http://www.natgeotraveller.in/six-years-and-counting/
- http://www.natgeotraveller.in/getting-saucy-about-food/
- http://www.natgeotraveller.in/what-dreams-may-come/
- http://www.natgeotraveller.in/train-to-nowhere/

2. PRE-PROCESSING DATA

- Importing required libraries

```
[1]  import nltk
     nltk.download("popular")
     nltk.download('stopwords')
     nltk.download('wordnet')
     nltk.download('punkt')
     nltk.download('averaged_perceptron_tagger')
```

- Import text file

From the links given above, text file is created on system and uploaded on python notebook.

Import text file

```
[2]  from google.colab import files
     uploaded = files.upload()
```

```
Choose Files  No file chosen    Upload widget is only available when the
Saving GettingSaucyAboutFood.txt to GettingSaucyAboutFood.txt
Saving SixYearsAndCounting.txt to SixYearsAndCounting.txt
Saving SixYearsAndCountingNew.txt to SixYearsAndCountingNew.txt
Saving TrainToNowhere.txt to TrainToNowhere.txt
Saving WhatDreamsMayCome.txt to WhatDreamsMayCome.txt
```

Note: Two text files uploaded are similar to test our model that whether it can detect the similarity between documents.

Now, we have to first read files for further pre processing

```
[18] doc_0 = open('GettingSaucyAboutFood.txt','r').read()
     doc_1 = open('SixYearsAndCounting.txt','r').read()
     doc_2 = open('TrainToNowhere.txt','r').read()
     doc_3 = open('WhatDreamsMayCome.txt','r').read()
     doc_4 = open('SixYearsAndCountingNew.txt','r').read()

     all_doc = [doc_0,doc_1,doc_2,doc_3,doc_4]
```

3. TOKENIZATION

We have performed word tokenization, so that later depending on tasks we can define our own conditions to divide the input texts into meaningful tokens.

Tokenization

```
[4]  #tokenising the text files
     from nltk import word_tokenize
     words0 = word_tokenize(doc_0)
     words1 = word_tokenize(doc_1)
     words2 = word_tokenize(doc_2)
     words3 = word_tokenize(doc_3)
     words4 = word_tokenize(doc_4)

     #Conversion to lower case for cosine similarities and future convenience
     print(words0,"\n",words1,"\n",words2,"\n",words3,"\n",words4,"\n")
     words0_new = [word.lower() for word in words0]
     words1_new = [word.lower() for word in words1]
     words2_new = [word.lower() for word in words2]
     words3_new = [word.lower() for word in words3]
     words4_new = [word.lower() for word in words4]
     print(words0_new,"\n",words1_new,"\n",words2_new,"\n",words3_new,"\n",words4_new)

     #tokeninzing with word boundaries may cause an issue so we can remove the punctuations
```

Now to improve the accuracy for the later evaluation metrics, we have removed the punctuations from the above text files.

```
[9]  text_p_0 = " ".join([char for char in words0_new if char not in string.punctuation])
     text_p_1 = " ".join([char for char in words1_new if char not in string.punctuation])
     text_p_2 = " ".join([char for char in words2_new if char not in string.punctuation])
     text_p_3 = " ".join([char for char in words3_new if char not in string.punctuation])
     text_p_4 = " ".join([char for char in words4_new if char not in string.punctuation])
```

And now we have tokenized again with text files without punctuations.

```
     #Tokenization after removing punctuation
     words0_t = word_tokenize(text_p_0)
     words1_t = word_tokenize(text_p_1)
     words2_t = word_tokenize(text_p_2)
     words3_t = word_tokenize(text_p_3)
     words4_t = word_tokenize(text_p_4)
     print(words0_t,"\n",words1_t,"\n",words2_t,"\n",words3_t,"\n",words4_t)
```

4. REMOVAL OF STOP WORDS

Stop words are basically those words which does not add any sense or meaning to the statement and hence can be ignored while reviewing a document.

Hence, we have removed all the stop words from the dataset.

```
# removing stopwords like i me myself we etc
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
print("The stopwords are",stop_words)
words0f = [word for word in words0_t if word not in stop_words]
words1f = [word for word in words1_t if word not in stop_words]
words2f = [word for word in words2_t if word not in stop_words]
words3f = [word for word in words3_t if word not in stop_words]
words4f = [word for word in words4_t if word not in stop_words]
print(words0f,"\n",words1f,"\n",words2f,"\n",words3f,"\n",words4f)
```

5. LEMMATIZATION

Lemmatization is the process where the model links the words with same meaning so that it can be analysed as a single item.

This also helps in getting similarities between two documents.

Lemmatization

```
[15] from nltk.stem import WordNetLemmatizer
     lemmatizer = WordNetLemmatizer()

     lem0 = [lemmatizer.lemmatize(word) for word in words0f]
     lem1 = [lemmatizer.lemmatize(word) for word in words1f]
     lem2 = [lemmatizer.lemmatize(word) for word in words2f]
     lem3 = [lemmatizer.lemmatize(word) for word in words3f]
     lem4 = [lemmatizer.lemmatize(word) for word in words4f]

     print(lem0,"\n",lem1,"\n",lem2,"\n",lem3,"\n",lem4)
```

6. STEMMING WITH POS TAGGING

Stemming is the process in which words of the document are reduced to its word stem basically removing suffixes and prefixes.

It is very much similar to lemmatization but they are not the same.

Example: ‘Caring’ lemmatization is ‘Care’ and Stemming is ‘Car’.

Stemming with POS Tagging

```
from nltk import LancasterStemmer
lc = LancasterStemmer()
stemmed0 = [lc.stem(word) for word in words0f]
stemmed1 = [lc.stem(word) for word in words1f]
stemmed2 = [lc.stem(word) for word in words2f]
stemmed3 = [lc.stem(word) for word in words3f]
stemmed4 = [lc.stem(word) for word in words4f]

print(stemmed0,"\n",stemmed1,"\n", stemmed2,"\n", stemmed3,"\n", stemmed4)
```

POS tagging is a task of labelling each word in a document with its appropriate part of speech.

```
[13] from nltk import pos_tag
     pos0 = pos_tag(stemmed0)
     pos1 = pos_tag(stemmed1)
     pos2 = pos_tag(stemmed2)
     pos3 = pos_tag(stemmed3)
     pos4 = pos_tag(stemmed4)
     print(pos0,"\n",pos1,"\n", pos2,"\n", pos3,"\n", pos4)
```

EVALUATION METRICS

7. LEXICAL RICHNESS

Sometimes lack of vocabulary can be an issue in assessing a document. So, by implementing lexical richness, the main idea is that if the text is more complex, we can use a varied vocabulary so that there's a large number of unique words.

Lexical Richness

```
!pip install lexicalrichness
```

```
[17] from lexicalrichness import LexicalRichness
```

Type-Token ratio:

TTR's Formula = Number of types divided by the number of tokens.

```
words = []
ttr = []
rttr = []
cttr = []
msttr = []
mattr = []
n = len(all_doc)
for i in range(n):
  lex = LexicalRichness(all_doc[i])
  print("Lexical Richness Assesment of text-", (i+1), " is:")
  print("Number of words: ", lex.words)
  words.append(lex.words)
  print("Type Token Ratio: ", lex.ttr)
  ttr.append(lex.ttr)
  print("Root type-token ratio: ", lex.rttr)
  rttr.append(lex.rttr)
  print("Corrected type-token ratio: ", lex.cttr, "\n")
  cttr.append(lex.cttr)

print("Final Insights: \nAverage number of words in the texts:", sum(words)/n,
      "\nAverage type-token ratio: ", sum(ttr)/n,
      "\nAverage root type-token ratio: ", sum(rttr)/n,
      "\nAverage corrected type-token ratio: ", sum(cttr)/n,
      )
```

8. READING INDEX

Reading Index (or Readability Index) involves the topic of determining the readability of a text. In general terms, this index indicates how difficult or easy it is to read or understand a text.

This metrics is performed on the tokenized text.

Reading Index

```
[20] from nltk.tokenize import sent_tokenize, word_tokenize
```

```
cont = 'yes'
vowels = ["a", "e", "i", "o", "u"]
endings = ["ed", "e", "es"]
reading_scores = {}

while(cont=="yes" or cont=="y"):
  name = input("Enter name of article: ")
  file_name = ''.join(name.title().split(" ")) + '.txt'
  try:
    article = open(file_name, 'r')
    text = article.read().lower()
    num_sentences = len(sent_tokenize(text))
    syllables = 0
    words = [word for word in word_tokenize(text) if word.isalpha()]
    num_words = len(words)
    for word in words:
      for vowel in vowels:
        syllables += word.count(vowel)
      for end in endings:
        if word.endswith(end) and (word.endswith('le')==False):
          syllables -= 1
    FRE = round(206.835 - (1.015*(num_words/num_sentences)) - (84.6 * (syllables/num_words)))
    reading_scores[name] = FRE
  except:
    print("Sorry this file does not exist in the database")
  cont = input("Any more articles to analyse? (yes-y,no-n): ")
  cont=cont.lower()
```

Finally, we can assess what is the reading difficulty of a doc or average reading difficulty of multiple docs.

```
[22] reading_scores
```

```
[25] avg_reading_score = 0
     for article in reading_scores:
       avg_reading_score += reading_scores[article]

     avg_reading_score = round(avg_reading_score/len(reading_scores))
     print(f"Average reading score: {avg_reading_score}")
```

9. COSINE SIMILARITY

Cosine Similarity is the process by which we can determine similarity index between two documents.

Mathematically, it measures the cosine of the angle between the vectors formed by the two text documents in a multi-dimensional space.

So, the smaller the angle, more is the value of cosine and hence we can determine the uniqueness of the document with respect to another document already in database.

```
def stem_tokens(tokens):
    return [stemmer.stem(item) for item in tokens]

def normalize(text):
    return stem_tokens(nltk.word_tokenize(text.lower().translate(remove_punctuation_map)))

vectorizer = TfidfVectorizer(tokenizer=normalize, stop_words='english')

def cosine_sim(text1, text2):
    tfidf = vectorizer.fit_transform([text1, text2])
    return ((tfidf * tfidf.T).A)[0,1]
```

```
[28] documents_instances_list = []
     for i in range(len(documents_list)):
       with open(documents_list[i]) as e:
         documents_instances_list.append(e.read())

[29] from itertools import combinations

     numbers = range(0, len(documents_instances_list))
     k = list(combinations(numbers, 2))
     m = lambda s: s.strip('./')
     document_map = dict(zip(numbers, list(map(m, documents_list))))
```

```
for i in range(len(k)):
  first_doc = k[i][0]
  second_doc = k[i][1]
  print("Document Similarity between document {}({}) and {}({}) is :
```

10. JACCARD SIMILARITY

Jaccard similarity is one of the most common used metrics in the field of NLP, as it basically scans a document to determine the level of duplicate detection.

It is measured as proportion of number of common words to number of unique words.

```
[49] from __future__ import division
     import string
     import math

     tokenize = lambda doc: doc.lower().split(" ")
```

```
[51] tokenized_documents = [tokenize(d) for d in all_doc] # tokenized docs
     all_tokens_set = set([item for sublist in tokenized_documents for item in sublist])
```

```
[52] def jaccard_similarity(query, document):
         intersection = set(query).intersection(set(document))
         union = set(query).union(set(document))
         return len(intersection)/len(union)
```

```
[69] for i in range(len(all_doc)):
       for j in range(i+1,len(all_doc)):
         x =  jaccard_similarity(tokenized_documents[i],tokenized_documents[j])
         print("Jaccard Similarity between doc_",i," and doc_",j," is :",x)
```

RESULT AND CONCLUSION

RESULTS:

- **Results of lexical richness on our dataset**

```
Lexical Richness Assesment of text- 1  is:
Number of words:  475
Type Token Ratio:  0.6610526315789473
Root type-token ratio:  14.407308087071279
Corrected type-token ratio:  10.187505247011888

Lexical Richness Assesment of text- 2  is:
Number of words:  463
Type Token Ratio:  0.6349892008639308
Root type-token ratio:  13.66333872280109
Corrected type-token ratio:  9.661439464541392

Lexical Richness Assesment of text- 3  is:
Number of words:  445
Type Token Ratio:  0.6449438202247191
Root type-token ratio:  13.605104792117345
Corrected type-token ratio:  9.620261857259768

Lexical Richness Assesment of text- 4  is:
Number of words:  409
Type Token Ratio:  0.6894865525672371
Root type-token ratio:  13.944002575442996
Corrected type-token ratio:  9.859898777978426

Lexical Richness Assesment of text- 5  is:
Number of words:  485
Type Token Ratio:  0.6329896907216495
Root type-token ratio:  13.940151902025544
Corrected type-token ratio:  9.85717594069281

Final Insights:
Average number of words in the texts: 455.4
Average type-token ratio:  0.6526923791912969
Average root type-token ratio:  13.911981215891652
Average corrected type-token ratio:  9.837256257496858
```

- **Results of reading index on our dataset**

```
[22] reading_scores

{'getting saucy about food': 54,
 'six years and counting': 47,
 'six years and counting new': 37,
 'train to nowhere': 63,
 'what dreams may come': 50}
```

```
Average reading score: 50
```

- **Results of cosine similarity index on our dataset**

```
Document Similarity between document 0(WhatDreamsMayCome.txt) and 1(SixYearsAndCountingNew.txt) is :     0.07257028953114779
Document Similarity between document 0(WhatDreamsMayCome.txt) and 2(GettingSaucyAboutFood.txt) is :     0.2914666725288135
Document Similarity between document 0(WhatDreamsMayCome.txt) and 3(SixYearsAndCounting.txt) is :      0.35512445823146854
Document Similarity between document 0(WhatDreamsMayCome.txt) and 4(TrainToNowhere.txt) is :     0.28698197023630667
Document Similarity between document 1(SixYearsAndCountingNew.txt) and 2(GettingSaucyAboutFood.txt) is :      0.06260830220535668
Document Similarity between document 1(SixYearsAndCountingNew.txt) and 3(SixYearsAndCounting.txt) is :   0.2417166612532536
Document Similarity between document 1(SixYearsAndCountingNew.txt) and 4(TrainToNowhere.txt) is :      0.06233000417541312
Document Similarity between document 2(GettingSaucyAboutFood.txt) and 3(SixYearsAndCounting.txt) is :   0.3521158847353581
Document Similarity between document 2(GettingSaucyAboutFood.txt) and 4(TrainToNowhere.txt) is :      0.35654439000382304
Document Similarity between document 3(SixYearsAndCounting.txt) and 4(TrainToNowhere.txt) is :   0.2929651043044028
```

- **Results of Jaccard similarity index on our dataset**

```
Jaccard Similarity between doc_ 0  and doc_ 1  is : 0.0817391304347826
Jaccard Similarity between doc_ 0  and doc_ 2  is : 0.09269162210338681
Jaccard Similarity between doc_ 0  and doc_ 3  is : 0.09124087591240876
Jaccard Similarity between doc_ 0  and doc_ 4  is : 0.07859531772575251
Jaccard Similarity between doc_ 1  and doc_ 2  is : 0.07927927927927927
Jaccard Similarity between doc_ 1  and doc_ 3  is : 0.08955223880597014
Jaccard Similarity between doc_ 1  and doc_ 4  is : 0.30103092783505153
Jaccard Similarity between doc_ 2  and doc_ 3  is : 0.08901515151515152
Jaccard Similarity between doc_ 2  and doc_ 4  is : 0.07241379310344828
Jaccard Similarity between doc_ 3  and doc_ 4  is : 0.07815275310834814
```

CONCLUSION:

After pre-processing the data, and evaluating it with different metrics that are available for assessing an individuality of the document. The results are average number of words in all texts is less than 500 and lexical richness shows average type-token ratio to be approx. 0.65 i.e. in a good range subject to quality of the text. Also, average reading index is equal to 50 which means that it is not very difficult to read and understand though, in general above 60 is a better range for positions like editor-in-chief.

Moving on next are two ways for checking similarity in documents. Since dataset is too small, cosine similarity index shows almost all the texts are unique with respect to each other but in case of Jaccard Similarity, there is an obvious observation that doc_1 and doc_4 is similar, since all other observations are less than 0.1.

Hence, out of 5 text files, 3 are unique i.e.

- GettingSaucyAboutFood.txt
- TrainToNowhere.txt
- WhatDreamsMayCome.txt

And 2 files are similar and can be considered as one i.e.

- SixYearsAndCounting.txt
- SixYearsAndCountingNew.txt

So, the uniqueness of the text files submitted by the shortlisted applicant is 80% and hence by that result, applicant may be hired.

CONTRIBUTION

Pre-processing data	Shounak, Navdeep
Cosine similarity	Shivam, Abhishek
Jaccard similarity and documentation	Sagar, Aniket
Lexical richness	Naynika
Reading index	Hrishita
Part of documentation	Preetham,Ritesh