Natural Language Processing (CSE4022)

NLP Hands on Activity

Name: Anmol Pant **Reg. No:** 18BCE0283

Slot: E1 + TE1 **Faculty:** Sharmila Banu K

Topic: National Geographic Traveler Article Analysis

Note:

The following document would contain my individual submission for the NatGeo traveler article similarity analysis hands on activity. Officially being a part of **Team 3**, the reason behind an individual submission would be the lack of coordination amidst the team members. We have had our share of problems being on the same page and delegating and distributing the tasks, and hence I decided to do it by myself and turn in an individual submission and I hope my case would be considered.

Approach and pipeline:

1. Extract text from the given corpus.
2. Perform stop word removal.
3. Tokenization
4. Formulate the bag of words representation.
5. Tf-idf vectorization
6. Compute cosine similarity and find document rank.
7. Compute Euclidian Distance and document rank.
8. Formulate results

The approach I have followed here is taking a random article of hers and assuming she has written the same for our publication, whose similarity we will be computing with the given 4 articles to get an idea of the 'originality' of her articles. The similarity metrics used for this purpose are cosine similarity and Euclidian distance, coupled with tf-idf vectorizer for vectorization and the bag of words representation to represent the words in the form of a dataframe.

Code Snippets and Screenshots:

Anmol Pant

18BCE0283

import libraries

In [1]:
```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

document extraction and stopword removal

In [2]:
```
def extract_doc(doc):
    f = open(doc,"r", encoding='utf-8')
    data = f.read()
    word_tokens = word_tokenize(data)
    stop_words = list(set(stopwords.words('english')))
    filtered_sentence = []
    stopwords_list = []
    for w in word_tokens:
        if w not in stop_words:
            filtered_sentence.append(w)
        else:
            stopwords_list.append(w)

    return filtered_sentence
```

In [3]:
```
d1 = extract_doc(r"C:\Users\anmol\Downloads\natgeo1.txt")
d2 = extract_doc(r"C:\Users\anmol\Downloads\natgeo2.txt")
d3 = extract_doc(r"C:\Users\anmol\Downloads\natgeo3.txt")
d4 = extract_doc(r"C:\Users\anmol\Downloads\natgeo4.txt")
d5 = extract_doc(r"C:\Users\anmol\Downloads\sample.txt")
```

bag of words representation

In [6]:
```
import pandas as pd

def Bag_of_words(word_list):
    word_l = []
    for word in distinct_words:
        if (word in word_list) == True:
            word_l.append(word_list.count(word))
        else:
            word_l.append(0)

    return word_l

df = pd.DataFrame()
df["words"] = distinct_words
df["d1"] = Bag_of_words(d1)
df["d2"] = Bag_of_words(d2)
df["d3"] = Bag_of_words(d3)
df["d4"] = Bag_of_words(d4)
df["d5"] = Bag_of_words(d5)


print(df.head(100))
```

```
          words  d1  d2  d3  d4  d5
0      server—a   0   0   1   0   0
1          meat   0   0   1   0   0
2             ;   0   0   1   2   1
3           buy   0   1   0   0   0
4          work   0   0   0   0   1
..          ...  ..  ..  ..  ..  ..
95  interesting   0   0   1   0   0
96        gates   0   0   0   0   1
97      manager   0   0   1   0   0
98    Traveller   1   0   0   0   0
99     drinking   0   1   0   0   0
```

tf-idf

In [9]:
```
import numpy as np
tfidf = pd.DataFrame()
tfidf["words"] = distinct_words

n1 = np.array(tf.d1)
n2 = np.array(tf.d2)
n3 = np.array(tf.d3)
n4 = np.array(tf.d4)
n5 = np.array(tf.d5)
x = np.array(idf.relevance)

tfidf["d1"] = n1*x
tfidf["d2"] = n2*x
tfidf["d3"] = n3*x
tfidf["d4"] = n4*x
tfidf["d5"] = n5*x

print(tfidf)
```

```
          words        d1        d2        d3        d4        d5
0       server-a  0.000000  0.000000  1.791759  0.000000  0.000000
1           meat  0.000000  0.000000  1.791759  0.000000  0.000000
2              ;  0.000000  0.000000  0.980829  1.497323  0.980829
3            buy  0.000000  1.791759  0.000000  0.000000  0.000000
4           work  0.000000  0.000000  0.000000  0.000000  1.791759
...          ...       ...       ...       ...       ...       ...
1062        Bolt  0.000000  0.000000  0.000000  1.791759  0.000000
1063     pretend  0.000000  0.000000  0.000000  1.791759  0.000000
1064        real  1.252763  1.252763  0.000000  0.000000  0.000000
1065        food  0.000000  0.980829  1.921577  0.000000  0.980829
1066  pugilistic  0.000000  0.000000  0.000000  1.791759  0.000000

[1067 rows x 6 columns]
```

Cosine Similarity

In [11]:
```
#cosine-similarity
import numpy as np
import math

cs = {}
q = 0
for i in np.array(tfidf.d5):
    q+=(i**2)

# print(q)

def cosine_sim(doc):
    d = 0
    for i in doc:
        d+=(i**2)

#     print(d)
    a = doc.dot(np.array(tfidf.d5))
#     print(a)
#     return (a/(math.sqrt(q*d)))
    return('{0:.10f}'.format((a/(math.sqrt(q*d)))))

cs["d1"] = float(cosine_sim(np.array(tfidf.d1)))
cs["d2"] = float(cosine_sim(np.array(tfidf.d2)))
cs["d3"] = float(cosine_sim(np.array(tfidf.d3)))
cs["d4"] = float(cosine_sim(np.array(tfidf.d4)))
print(cs)
```

```
{'d1': 0.0698268963, 'd2': 0.0472304297, 'd3': 0.0524115398, 'd4': 0.0783768587}
```

Eucledian Distance and Document Ranking

In [15]:
```
ed = {}
def euc_d(doc,query):
    e = 0
    for i in range(len(doc)):
        e+=(doc[i] - query[i])**2

    return ('{0:.10f}'.format(math.sqrt(e)))

ed["d1"] = float(euc_d(np.array(tfidf.d1), np.array(tfidf.d5)))
ed["d2"] = float(euc_d(np.array(tfidf.d2), np.array(tfidf.d5)))
ed["d3"] = float(euc_d(np.array(tfidf.d3), np.array(tfidf.d5)))
ed["d4"] = float(euc_d(np.array(tfidf.d4), np.array(tfidf.d5)))
print(ed)
print(list(ed.values()))

def rank_doc(array):
    l = array.copy()
    valid = True
    while valid:
        if(len(np.where(array == min(l))[0]) > 1):
            for i in range(len(np.where(array == min(l))[0])):
                print((np.where(array == min(l))[0][i]+1),)
            l = np.delete(l,np.where(l == min(l)),0)
        else:
            print(np.where(array == min(l))[0][0]+1)
            l = np.delete(l,np.where(l == min(l))[0],0)

        if(len(l) == 0):
            valid = False


rank_doc(np.array(list(ed.values())))
```

```
{'d1': 0.0342566834, 'd2': 0.0340484218, 'd3': 0.0357740865, 'd4': 0.0338834713}
[0.0342566834, 0.0340484218, 0.0357740865, 0.0338834713]
4
2
1
3
```

Final Note:

We see that after computing the **cosine similarity** and **Euclidean distance** between the sample article and the 4 articles given to us in the form of the corpus, the similarity is almost negligible. Hence it is safe to assume that the articles she writes are original and we can hire her for our publication.

Since enclosing all the code snippets is beyond the scope of this document, I have only attached the prominent and the most important ones here, the entirety of this jupyter notebook can be checked out at:

https://drive.google.com/drive/folders/1kfh_iSe3P6bNbf_YWxp4qZVEB3EXUqhJ

***