WEB MINING

[ASSIGNMENT-2]

Roll no.-2081001204
M.Tech. (2nd Semester)
(Statistical Computing-Data Science)
Submitted by: Anjali Gautam

Ques. Take some documents from Google by firing queries. You need to cluster the results. Take 4 queries and take top m documents for each query. Combine all results. m can be different for different queries.
Extract the main contents of documents from html file. Do required preprocessing and make term document matrix Using tfidf.
Cluster the matrices using K means and cosine similarity. K = number of queries.
We expect each cluster as representative of one query each.
Comment on the result.
Evaluate it using intracluster and intercluster distance. Evaluate using benchmark as it is known to you (if this evaluation seems reasonable i.e. you are able to interpret clusters .

- ***Import required packages such as re, pandas, numpy, nltk, TfIdfVectorizer etc.to perform the functionality.***

```
import urllib.request
from urllib.request import urlopen
from bs4 import BeautifulSoup
from inscriptis import get_text
import re
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.cluster import KMeans
```

- ***PorterStemmer is for finding root/stem of word in document***

```
ps = PorterStemmer()
```

- ***Take 4 queries and take top m documents for each query using google search and combine all the documents.***

```
combined_queries=[]
try:
  from googlesearch import search
except ImportError:
    print("No module named 'google' found")
# to search
query1=input("enter the first query")
for j in search(query1, tld="co.in", num=10, stop=2, pause=2):
  print(j)
  combined_queries.append(j)
query2=input("enter the second query")
for k in search(query2, tld="co.in", num=10, stop=3, pause=2):
  print(k)
  combined_queries.append(k)
query3=input("enter the third query")
for l in search(query3, tld="co.in", num=10, stop=2, pause=2):
  print(l)
  combined_queries.append(l)
query4=input("enter the fourth query")
```

```
for m in search(query4, tld="co.in", num=10, stop=2, pause=2):
   print(m)
   combined_queries.append(m)
print(combined_queries)
```

- ***Extract the all html documents from the combined documents.***

```
html_doc=[]
for ele in combined_queries:
   result=ele.endswith('.html')
   result1=ele.endswith('.htm')
   if result== True or result1==True:
      html_doc.append(ele)
print(html_doc)
print('total html documents  ',len(html_doc))
```

- ***Parsing of all the extracted HTML documents.***

```
all_text=[]
for url in html_doc:
   #To open the url
   html = urlopen(url)
   #Use html.parser to parse the html file
   bs = BeautifulSoup(html, 'html.parser')
   html1 = urllib.request.urlopen(url).read().decode('utf-8')
   #For getting the text from the Html file
   text = get_text(html1)
   all_text.append(text)
print(all_text)
```

- ***Pre-processing of all above parsed HTML Documents***

```
wordnet=WordNetLemmatizer()
def preprocess(set_of_doc):
   corpus = []
   for i in set_of_doc:
      qs=cleanpunc(i)
      qs=cleanhtml(qs)
      review = re.sub('[^a-zA-Z]', ' ',qs)
      review = review.lower()
      review = review.split()
      review = [wordnet.lemmatize(word) and ps.stem(word) for word in review if not word
in set(stopwords.words('english'))]
      review = ' '.join(review)
      corpus.append(review)
   return corpus
def cleanpunc(sentenc):
   cleaned=re.sub(r'[?|!|\'|"|#]',r'',sentenc)
```

```
    cleaned=re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return cleaned
def cleanhtml(sentence):
    cleanr=re.compile('<.*?>')
    cleantext= re.sub(cleanr,' ',sentence)
    return cleantext
preprocessed_doc=preprocess(all_text)
print(preprocessed_doc)
```

- ***Make term document matrix Using tfidf.***

```
vectorizer = TfidfVectorizer()
doc_vec = vectorizer.fit_transform(preprocessed_doc)
df=pd.DataFrame(doc_vec.toarray(), columns=vectorizer.get_feature_names())
print(df)
```

- ***To Cluster the matrices using K means and cosine similarity where K = number of queries.***

```
number_clusters = 4
km = KMeans(n_clusters=number_clusters)
km.fit(doc_vec)
print("Top terms per cluster:")
order_centroids = km.cluster_centers_.argsort()[:, ::-1]
terms = vectorizer.get_feature_names()
for i in range(number_clusters):
   top_ten_words = [terms[ind] for ind in order_centroids[i, :5]]
   print("Cluster {}: {}".format(i, ' '.join(top_ten_words)))
```