Digital Assignment 1

Devang Mehrotra

18BCE0763

Q1-

a) Extract the source content (excluding any tags) from the website (https://en.wikipedia.org/wiki/Web_mining).

```
In [3]: import re
        from bs4 import BeautifulSoup
        from urllib import request

In [5]: url = "https://en.wikipedia.org/wiki/Web_mining"
        html = request.urlopen(url).read().decode('utf8')
        raw = BeautifulSoup(html, 'html.parser').get_text()

In [7]: print(raw)
Finding extract rules
Finding patterns in text


Finding frequent sub structures
Web site schema discovery


Categorization
Clustering


Site construction
Adaptation and management

Web usage mining[edit]
Web usage mining is the application of data mining techniques to discover interesting usage patterns from Web data in order t
o understand and better serve the needs of Web-based applications.
Usage data captures the identity or origin of Web users along with their browsing behavior at a Web site.
Web usage mining itself can be classified further depending on the kind of usage data considered:
```

b) Display the total number of terms and term frequency of each term present in them after applying stop word removal.

Total words-

```
In [9]: from nltk.corpus import stopwords
        from nltk.tokenize import word_tokenize
        stop_words = set(stopwords.words('english'))
        word_tokens = word_tokenize(raw)
        filtered_sentence = [w for w in word_tokens if not w in stop_words]

        filtered_sentence = []

        for w in word_tokens:
            if w not in stop_words:
                filtered_sentence.append(w)
        print("total terms=",len(filtered_sentence))

        total terms= 2996
```

Term Frequency-

```
In [26]: d = {}

for i in filtered_sentence:
    if i in d:
        d[i] += 1
    else:
        d[i] = 1
for key in list(d.keys()):
     print(key, ":", d[key])
```

```
Web : 100
mining : 74
- : 3
Wikipedia : 8
document.documentElement.className= : 1
'' : 503
client-js : 1
; : 50
RLCONF= : 1
{ : 45
`` : 59
wgBreakFrames : 1
: : 183
! : 18
1 : 20
, : 420
wgSeparatorTransformTable : 1
[ : 40
] : 40
```

c) Remove all the special characters/symbols present in the content by adding those characters as stopwords in the existing stopword list.

```
In [12]: STOP_WORDS = set(stopwords.words('english'))
STOP_WORDS.add('-')
STOP_WORDS.add(',')
STOP_WORDS.add("\'s")
STOP_WORDS.add("\'")
STOP_WORDS.add('.')
STOP_WORDS.add(',')
STOP_WORDS.add('(')
STOP_WORDS.add(')')
STOP_WORDS.add('[')
STOP_WORDS.add(']')
STOP_WORDS.add(';')
STOP_WORDS.add(':')
STOP_WORDS.add('^')
STOP_WORDS.add('..')
STOP_WORDS.add('&')
STOP_WORDS.add('=')
STOP_WORDS.add('?')

word_tokens = word_tokenize(raw)
filtered_sentence1 = [w for w in word_tokens if not w in STOP_WORDS]
print("total words after removing special char=",len(filtered_sentence1))
```

```
total words after removing special char= 2389
```

d) Also, apply stemming (don't use porter stemmer) and lemmatization to the same document and display the number of terms along with their corresponding stemmed as well as lemmatized words present in them.

```
In [30]: from nltk.stem.snowball import SnowballStemmer
from nltk.stem import WordNetLemmatizer
ps = SnowballStemmer("english")
lemmatizer = WordNetLemmatizer()
stems = []
lemma = []
for w in filtered_sentence1:
    print(w,"-",ps.stem(w), " - ", lemmatizer.lemmatize(w))
    stems.append(ps.stem(w))
    lemma.append(lemmatizer.lemmatize(w))
```

In [31]:
```
data={"Original Term":filtered_sentence1,"Stemmed Term":stems,"Lemmatized Term":lemma}
import pandas
pandas.DataFrame(data)
```

Out[31]:

	Original Term	Stemmed Term	Lemmatized Term
0	Web	web	Web
1	mining	mine	mining
2	Wikipedia	wikipedia	Wikipedia
3	document.documentElement.className=	document.documentelement.classname=	document.documentElement.className=
4	''	''	''
...	...	...	...
3602	''	''	''
3603	mw1409	mw1409	mw1409
3604	''	''	''
3605	)	)	)
3606	)	)	)

3607 rows × 3 columns

e) Count the total number of stemmed and lemmatized words.

In [33]:
```
frequency1 = {}
for word in stems:
    count = frequency1.get(word,0)
    frequency1[word] = count + 1
frequency_list1 = frequency1.keys()
print("stemmed words-",len(frequency_list1))
```

```
stemmed words- 1226
```

In [34]:
```
frequency2 = {}
for word in lemma:
    count = frequency2.get(word,0)
    frequency2[word] = count + 1
frequency_list2 = frequency2.keys()
print("lemmatized words-",len(frequency_list2))
```

```
lemmatized words- 1435
```

f) Display the POS tag (sentence-wise) for all the stopwords (excluding the special character/symbols), which are removed from the content, using pandas dataframe

In [36]:
```
import nltk
nltk.download('averaged_perceptron_tagger')
finalwords = []

for line in filtered_sentence1:
        for word in line.split():
            finalwords.append(word)
tagged = nltk.pos_tag(finalwords)
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\Sreemanth\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

In [37]:
```
import pandas as pd
df = pd.DataFrame(tagged,columns=['List of Stopwords','POS Tags'])
print(df)
```

```
                        List of Stopwords POS Tags
0                                     Web      NNP
1                                  mining       NN
2                               Wikipedia      NNP
3     document.documentElement.className=       NN
4                                      ''       ''
...                                   ...      ...
3602                                   ''       ''
3603                                mw1409       NN
3604                                   ''       ''
3605                                    }        )
3606                                    }        )

[3607 rows x 2 columns]
```

Q2-

a) Extract the contents (excluding any tags) from two websites (https://en.wikipedia.org/wiki/Web_mining&https://en.wikipedia.org/wiki/Data_ mining)

In [1]:
```
from urllib import request
import re
```

In [2]:
```
r1=request.urlopen('https://en.wikipedia.org/wiki/Web_mining').read().decode('utf8')
```

In [3]:
```
from bs4 import BeautifulSoup
soup1 = BeautifulSoup(r1,'html.parser')
```

In [4]:
```
text1 = soup1.find('p')
text1.get_text()
# soup1 = BeautifulSoup(resulttext,'html.parser').get_text()
```

Out[4]:
```
'Web mining is the application of data mining techniques to discover patterns from the World Wide Web. As the name proposes, th
is is information gathered by mining the web. It makes utilization of automated apparatuses to reveal and extricate data from s
ervers and web2 reports, and it permits organizations to get to both organized and unstructured information from browser activi
ties, server logs, website and link structure, page content and different sources.\n'
```

In [5]:
```
r2=request.urlopen('https://en.wikipedia.org/wiki/Data_mining').read().decode('utf8')
from bs4 import BeautifulSoup
soup2 = BeautifulSoup(r2,'html.parser')
text2 = soup2.find('p')
text2.get_text()
```

Out[5]:
```
'Data mining is a process of discovering patterns in large data sets involving methods at the intersection of machine learning,
statistics, and database systems.[1] Data mining is an interdisciplinary subfield of computer science and statistics with an ov
erall goal to extract information (with intelligent methods) from a data set and transform the information into a comprehensibl
e structure for further use.[1][2][3][4] Data mining is the analysis step of the "knowledge discovery in databases" process, or
KDD.[5] Aside from the raw analysis step, it also involves database and data management aspects, data pre-processing, model and
inference considerations, interestingness metrics, complexity considerations, post-processing of discovered structures, visuali
zation, and online updating.[1]\n'
```

b) Remove stopwords (including the special characters/symbols) from the contents retrieved from those two URLs and save the contents in two separate .doc file.

In [6]:
```
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import sent_tokenize, word_tokenize
import pandas as pd
import re
stop_words = (stopwords.words('english'))
```

In [7]:
```
stop_words.append('.')
stop_words.append(',')
stop_words.append('/')
stop_words.append('!')
stop_words.append('@')
stop_words.append('#')
stop_words.append('$')
stop_words.append('%')
stop_words.append('^')
stop_words.append('&')
stop_words.append('*')
stop_words.append('(')
stop_words.append(')')
stop_words.append('_')
stop_words.append('-')
stop_words.append('+')
stop_words.append('=')
stop_words.append('[')
stop_words.append(']')
stop_words.append(';')
stop_words.append('{')
stop_words.append('}')
stop_words.append('~')
stop_words.append('`')
stop_words.append('``')
stop_words.append("''")
word_tokens1 = word_tokenize(text1.get_text())
word_tokens2 = word_tokenize(text2.get_text())
```

In [10]:
```
words1 = [w for w in word_tokens1 if not w in stop_words]
words2 = [w for w in word_tokens2 if not w in stop_words]
```

In [11]:
```
words1,words2
```

```
 'discover',
 'patterns',
 'World',
 'Wide',
 'Web',
 'As',
 'name',
 'proposes',
 'information',
 'gathered',
 'mining',
 'web',
 'It',
 'makes',
 'utilization',
 'automated',
 'apparatuses',
 'reveal',
 'extricate',
 'data',
```

In [14]:
```
with open('doc1.doc', 'w') as f:
    for item in words1:
        f.write("%s\n" % item)
with open('doc2.doc', 'w') as f:
    for item in words2:
        f.write("%s\n" % item)
```

In [15]:
```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
```

In [16]:
```
readwords1 = []
readwords2 = []
with open('doc1.doc','r') as file:
    for line in file:
        for word in line.split():
            readwords1.append(word)
with open('doc2.doc','r') as file:
    for line in file:
        for word in line.split():
            readwords2.append(word)
```

In [17]:
```
text1 = " ".join(readwords1)
text2 = " ".join(readwords2)
```

In [18]:
```
text1
```

Out[18]:
```
'Web mining application data mining techniques discover patterns World Wide Web As name proposes information gathered mining we
b It makes utilization automated apparatuses reveal extricate data servers web2 reports permits organizations get organized uns
tructured information browser activities server logs website link structure page content different sources'
```

In [19]:
```
text2
```

Out[19]:
```
'Data mining process discovering patterns large data sets involving methods intersection machine learning statistics database s
ystems 1 Data mining interdisciplinary subfield computer science statistics overall goal extract information intelligent method
s data set transform information comprehensible structure use 1 2 3 4 Data mining analysis step knowledge discovery databases p
rocess KDD 5 Aside raw analysis step also involves database data management aspects data pre-processing model inference conside
rations interestingness metrics complexity considerations post-processing discovered structures visualization online updating
1'
```

c) Display the Term-Document incidence matrix using Boolean, Bag-of-words and Complete representation (Use pandas dataframe)

In [91]:
```
booldoc1 = []
booldoc2 = []
for x in (text1 + text2).split(" "):
    if(x in text1.split(" ")):
        booldoc1.append(1)
    else:
        booldoc1.append(0)
for x in (text1 + text2).split(" "):
    if(x in text2.split(" ")):
        booldoc2.append(1)
    else:
        booldoc2.append(0)
booldata = {"Words":(text1 + text2).split(" "),"Doc 1":booldoc1,"Doc 2":booldoc2}
pd.DataFrame(booldata)
```

Out[91]:

	Words	Doc 1	Doc 2
0	Web	1	0
1	mining	1	1
2	application	1	0
3	data	1	1
4	mining	1	1
...	...	...	...
117	structures	0	1
118	visualization	0	1
119	online	0	1
120	updating	0	1
121	1	0	1

122 rows × 3 columns

In [21]:
```
vec = CountVectorizer()
```

In [53]:
```
X = vec.fit_transform(docs)
X = X.transpose()
```

In [60]:
```
#df =pd.DataFrame(X.toarray(), columns=vec.get_feature_names())
#pd.DataFrame(X.toarray(), columns=vec.get_feature_names())
df =pd.DataFrame(X.toarray(), columns=['doc1 Freq','doc2 Freq'])
df['words'] = vec.get_feature_names()
```

In [64]:
```
#print(df)
df[['words','doc1 Freq','doc2 Freq']]
```

Out[64]:

	words	doc1 Freq	doc2 Freq
0	activities	1	0
1	also	0	1
2	analysis	0	2
3	apparatuses	1	0
4	application	1	0
...	...	...	...
85	web	3	0
86	web2	1	0
87	website	1	0
88	wide	1	0
89	world	1	0

90 rows × 3 columns

In [83]:
```
compwords1 =[]
pos1=[]
for x in resulttext1.get_text().split(" "):
    m =0
    if x in words1:
        for y in text1.split(" "):
            m += 1
            if(x==y):
                compwords1.append(x)
                pos1.append(text1.index(x,m-1))
finaldata1 = {"words":compwords1,"Position":pos1}
pd.DataFrame(finaldata1)
```

Out[83]:

	words	Position
0	Web	0
1	Web	75
2	mining	4
3	mining	4
4	mining	28
5	application	11
6	data	23
7	data	188
8	mining	4
9	mining	4
10	mining	28
11	techniques	35
12	discover	46
13	patterns	55
14	World	64

d) Input a search a query (preferably a sentence) and compare the contents of the both pages with the processed query. Display the similarity result based on highest frequency matching count of the term.

In [115]:
```
searchstring = "Web hello intersection data is analysis interdisciplinary subfield usefull"

sc1=0
sc2=0
for x in searchstring.split(" "):
        if(x in text1.split(" ")):
            sc1=sc1+1
        if(x in text2.split(" ")):
            sc2=sc2+1

if(sc1>sc2):
    print("Doc1 is more relevent")
elif(sc1==sc2):
    print("Both Documents are equally relevent")
elif(sc1<sc2):
    print("Doc2 is more relevent")
```

```
Doc2 is more relevent
```

In [84]:
```
compwords2 =[]
pos2=[]
for x in resulttext1.get_text().split(" "):
    m =0
    if x in words2:
        for y in text2.split(" "):
            m += 1
            if(x==y):
                compwords2.append(x)
                pos2.append(text2.index(x,m-1))
finaldata2 = {"words":compwords2,"Position":pos2}
pd.DataFrame(finaldata2)
```

Out[84]:

	words	Position
0	mining	5
1	mining	140
2	mining	140
3	data	47
4	data	47
5	data	116
6	data	116
7	mining	5
8	mining	140
9	mining	140
10	patterns	32
11	information	223
12	information	223

Q3-

Write a python program to prepare the Word Clouds representation based on the content present in the two document files prepared in Q.No. 2

In [2]:
```
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS
```

In [3]:
```
def random_color_func(word=None, font_size=None, position=None, orientation=None, font_path=None, random_state=None):
    h = int(360.0 * 45.0 / 255.0)
    s = int(100.0 * 255.0 / 255.0)
    l = int(100.0 * float(random_state.randint(60, 120)) / 255.0)

    return "hsl({}, {}%, {}%)".format(h, s, l)
```

In [9]:
```
file_content1=open ("doc1.doc").read()
file_content2=open ("doc1.doc").read()
```

In [10]:
```
wordcloud1 = WordCloud(font_path = r'C:\Windows\Fonts\Verdana.ttf',
                       stopwords = STOPWORDS,
                       background_color = 'white',
                       width = 1200,
                       height = 1000,
                       color_func = random_color_func
                       ).generate(file_content1)
wordcloud2 = WordCloud(font_path = r'C:\Windows\Fonts\Verdana.ttf',
                       stopwords = STOPWORDS,
                       background_color = 'white',
                       width = 1200,
                       height = 1000,
                       color_func = random_color_func
                       ).generate(file_content2)
```

In [11]:
```
plt.imshow(wordcloud1)
plt.axis('off')
plt.show()
```

In [12]:
```
plt.imshow(wordcloud2)
plt.axis('off')
plt.show()
```

Q4-

Write a python program to show the implementation of sentence paraphrasing through synonyms (retaining semantic meaning) for the following four sentences. Display at least three other paraphrased sentences for each sentence mentioned below.

a. The quick brown fox jumps over the lazy dog

b. Obama and Putin met the previous week

c. At least 12 people were killed in the battle last week

d. I will go home and come back tomorrow.

In [11]:
```
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
from nltk.corpus import wordnet as wn
import pandas as pd
def tag(sentence):
 words = word_tokenize(sentence)
 words = pos_tag(words)
 return words

def paraphraseable(tag):
 return tag.startswith('NN') or tag == 'VB' or tag.startswith('JJ')

def pos(tag):
 if tag.startswith('NN'):
  return wn.NOUN
 elif tag.startswith('V'):
  return wn.VERB

def synonyms(word, tag):
    lemma_lists = [ss.lemmas() for ss in wn.synsets(word, pos(tag))]
    lemmas = [lemma.name() for lemma in sum(lemma_lists, [])]
    return set(lemmas)

def synonymIfExists(sentence):
 for (word, t) in tag(sentence):
   if paraphraseable(t):
    syns = synonyms(word, t)
    if syns:
     if len(syns) > 1:
      yield [word, list(syns)]
      continue
   yield [word, []]

def paraphrase(sentence):
 return [x for x in synonymIfExists(sentence)]
```

In [31]:
```
s1 = paraphrase("The quick brown fox jumps over the lazy dog")
s2 = paraphrase("Obama and Putin met the previous week")
s3 = paraphrase("At least 12 people were killed in the battle last week")
s4 = paraphrase("I will go home and come back tomorrow")
```

In [32]:
```
s1,s2,s3,s4
```

Out[32]:
```
([['The', []],
  ['quick',
   ['fast',
    'speedy',
    'prompt',
    'spry',
    'agile',
    'immediate',
    'quickly',
    'nimble',
    'promptly',
    'quick',
    'straightaway',
    'flying',
    'warm',
    'ready']],
  ['brown',
   ['Brown_University',
    'brownness',
```

In [58]:
```
def parap(arr,j):
    s1 = ""

    for i in range(len(arr)):
        if(arr[i][1]==[]):
            wrd=arr[i][0]
        elif(len(arr[i][1])<=j):
            wrd=arr[i][0]
        else:
            wrd=arr[i][1][j]

        s1=s1+" "+wrd
    return s1
```

In [59]:
```
parap(s1,1)
```

Out[59]: ' The speedy brownness Charles_James_Fox jumps over the otiose bounder'

In [60]:
```
parap(s1,2)
```

Out[60]: ' The prompt Brown George_Fox jumps over the lazy dog'

In [61]:
```
parap(s1,3)
```

Out[61]: ' The spry Robert_Brown fox jumps over the indolent frank'

In [62]:
```
parap(s2,1)
```

Out[62]: ' Obama and Vladimir_Putin met the late calendar_week'

In [63]:
```
parap(s2,2)
```

Out[63]: ' Obama and Vladimir_Vladimirovich_Putin met the former hebdomad'

In [64]:
```
parap(s2,3)
```

Out[64]: ' Obama and Putin met the premature week'

In [65]:
```
parap(s3,1)
```

Out[65]: ' At to_the_lowest_degree 12 hoi_polloi were killed in the struggle lastly calendar_week'

In [66]:
```
parap(s3,2)
```

Out[66]: ' At least 12 multitude were killed in the battle final_stage hebdomad'

In [68]:
```
parap(s3,3)
```

Out[68]: " At least 12 masses were killed in the fight shoemaker's_last week"

In [69]:
```
parap(s4,1)
```

Out[69]: ' I will rifle house and fare back tomorrow'

In [70]:
```
parap(s4,2)
```

Out[70]: ' I will run nursing_home and total back tomorrow'

In [71]:
```
parap(s4,3)
```

Out[71]: ' I will proceed base and occur back tomorrow'

In [73]:
```
for i in range(3):
    print(parap(s1,i))
```

```
The fast Brown_University slyboots jumps over the faineant detent
The speedy brownness Charles_James_Fox jumps over the otiose bounder
The prompt Brown George_Fox jumps over the lazy dog
```

In [74]:
```
for i in range(3):
    print(parap(s2,i))
```

```
Obama and Putin met the previous workweek
Obama and Vladimir_Putin met the late calendar_week
Obama and Vladimir_Vladimirovich_Putin met the former hebdomad
```

In [75]:
```
for i in range(3):
    print(parap(s3,i))
```

```
At least 12 citizenry were killed in the engagement death workweek
At to_the_lowest_degree 12 hoi_polloi were killed in the struggle lastly calendar_week
At least 12 multitude were killed in the battle final_stage hebdomad
```

In [76]:
```
for i in range(3):
    print(parap(s4,i))
```

```
I will belong plate and come back tomorrow
I will rifle house and fare back tomorrow
I will run nursing_home and total back tomorrow
```