Topic Modeling

In [1]:

```
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
import string
```

In [2]:

```
doc1 = "Sugar is bad to consume. My sister likes to have sugar, but not my father."
doc2 = "My father spends a lot of time driving my sister around to dance practice."
doc3 = "Doctors suggest that driving may cause increased stress and blood pressure."
doc4 = "Sometimes I feel pressure to perform well at school, but my father never seems to d
doc5 = "Health experts say that Sugar is not good for your lifestyle."

# compile documents
doc_complete = [doc1, doc2, doc3, doc4, doc5]
```

In [3]:

```
stop = set(stopwords.words('english'))
exclude = set(string.punctuation)
lemma = WordNetLemmatizer()
```

In [4]:

```
def clean(doc):
    stop_free = " ".join([i for i in doc.lower().split() if i not in stop])
    punc_free = ''.join(ch for ch in stop_free if ch not in exclude)
    normalized = " ".join(lemma.lemmatize(word) for word in punc_free.split())
    return normalized
```

In [5]:

```
import nltk
#nltk.download('wordnet')
```

In [6]:

```
doc_clean = [clean(doc).split() for doc in doc_complete]
```

In [7]:

```
# Importing Gensim
import gensim
from gensim import corpora
```

In [8]:

```
# Creating the term dictionary of our courpus, where every unique term is assigned an index
dictionary = corpora.Dictionary(doc_clean)
```

In [9]:

```
# Converting list of documents (corpus) into Document Term Matrix using dictionary prepared
doc_term_matrix = [dictionary.doc2bow(doc) for doc in doc_clean]
```

In [10]:

```
# Creating the object for LDA model using gensim library
Lda = gensim.models.ldamodel.LdaModel
```

In [11]:

```
# Running and Trainign LDA model on the document term matrix.
ldamodel = Lda(doc_term_matrix, num_topics=2, id2word = dictionary, passes=50)
```

In [12]:

```
print(ldamodel.print_topics(num_topics=2, num_words=2))
```

```
[(0, '0.060*"driving" + 0.036*"blood"'), (1, '0.070*"sugar" + 0.069*"fathe
r"')]
```

In [ ]:

```

```

Aspect Mining

In [3]:

```
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
from nltk.corpus import wordnet
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.stem.wordnet import WordNetLemmatizer
import stanfordnlp
import stanza
```

In [4]:

```
#pip install stanza
```

In [5]:

```
# Make sure you have downloaded the StanfordNLP English model and other essential tools usi
#stanfordnlp.download('en')
#nltk.download('stopwords')
#nltk.download('punkt')
#nltk.download('averaged_perceptron_tagger')
#stanza.download('en')
```

In [6]:

```
txt = "The Sound Quality is great but the battery life is very bad."
```

In [7]:

```
txt = txt.lower()   # LowerCasing the given Text
sentList = nltk.sent_tokenize(txt)  # Splitting the text into sentences
```

In [8]:

```
fcluster = []
totalfeatureList = []
finalcluster = []
categories = []
dic = {}
```

In [9]:

```
for line in sentList:
    txt_list = nltk.word_tokenize(line)    # Splitting up into words
    taggedList = nltk.pos_tag(txt_list)    # Doing Part-of-Speech Tagging to each word
print(taggedList)
```

```
[('the', 'DT'), ('sound', 'NN'), ('quality', 'NN'), ('is', 'VBZ'), ('great',
'JJ'), ('but', 'CC'), ('the', 'DT'), ('battery', 'NN'), ('life', 'NN'), ('i
s', 'VBZ'), ('very', 'RB'), ('bad', 'JJ'), ('.', '.')]
```

In [10]:

```
newwordList = []
flag = 0
for i in range(0,len(taggedList)-1):
    if(taggedList[i][1]=="NN" and taggedList[i+1][1]=="NN"):     # If two consecutive words
        newwordList.append(taggedList[i][0]+taggedList[i+1][0])
        flag=1
    else:
        if(flag==1):
            flag=0
            continue
        newwordList.append(taggedList[i][0])
        if(i==len(taggedList)-2):
            newwordList.append(taggedList[i+1][0])
finaltxt = ' '.join(word for word in newwordList)
print(finaltxt)
```

```
the soundquality is great but the batterylife is very bad .
```

In [11]:

```
stop_words = set(stopwords.words('english'))
new_txt_list = nltk.word_tokenize(finaltxt)
wordsList = [w for w in new_txt_list if not w in stop_words]
taggedList = nltk.pos_tag(wordsList)
```

In [12]:

```
nlp = stanza.Pipeline('en') # initialize English neural pipeline
doc = nlp(finaltxt)         # Object of Stanford NLP Pipeleine

# Getting the dependency relations betwwen the words
dep_node = []
for dep_edge in doc.sentences[0].dependencies:
 dep_node.append([dep_edge[2].text, dep_edge[0].id, dep_edge[1]])

# Coverting it into appropriate format
for i in range(0, len(dep_node)):
 if (int(dep_node[i][1]) != 0):
  dep_node[i][1] = newwordList[(int(dep_node[i][1]) - 1)]
#print(dep_node)
```

```
2022-11-21 00:12:43 INFO: Checking for updates to resources.json in case mod
els have been updated.  Note: this behavior can be turned off with download_
method=None or download_method=DownloadMethod.REUSE_RESOURCES
```

Downloading	193k/?
https://raw.githubusercontent.com/stanfordnlp/stanza-	[00:00<00:00,
resources/main/resources_1.4.1.json:	5.50MB/s]

```
2022-11-21 00:12:45 INFO: Loading these models for language: en (English):
============================
| Processor    | Package   |
----------------------------
| tokenize     | combined  |
| pos          | combined  |
| lemma        | combined  |
| depparse     | combined  |
| sentiment    | sstplus   |
| constituency | wsj       |
| ner          | ontonotes |
============================

2022-11-21 00:12:45 INFO: Use device: cpu
2022-11-21 00:12:45 INFO: Loading: tokenize
2022-11-21 00:12:45 INFO: Loading: pos
2022-11-21 00:12:46 INFO: Loading: lemma
2022-11-21 00:12:46 INFO: Loading: depparse
2022-11-21 00:12:46 INFO: Loading: sentiment
2022-11-21 00:12:46 INFO: Loading: constituency
2022-11-21 00:12:47 INFO: Loading: ner
2022-11-21 00:12:47 INFO: Done loading processors!
```

In [13]:

```
featureList = []
categories = []
for i in taggedList:
    if(i[1]=='JJ' or i[1]=='NN' or i[1]=='JJR' or i[1]=='NNS' or i[1]=='RB'):
        featureList.append(list(i))          # For features for each sentence
        totalfeatureList.append(list(i))     # This list will store all the features for eve
        categories.append(i[0])
print(featureList)
#print(categoriesList)
```

```
[['soundquality', 'NN'], ['great', 'JJ'], ['batterylife', 'NN'], ['bad', 'J
J']]
```

In [14]:

```
fcluster = []
for i in featureList:
    filist = []
    for j in dep_node:
        if((j[0]==i[0] or j[1]==i[0]) and (j[2] in ["nsubj", "acl:relcl", "obj", "dobj", "a
            if(j[0]==i[0]):
                filist.append(j[1])
            else:
                filist.append(j[0])
    fcluster.append([i[0], filist])
print(fcluster)
```

```
[['soundquality', ['great']], ['great', ['soundquality']], ['batterylife',
['bad']], ['bad', ['batterylife', 'very']]]
```

In [15]:

```
finalcluster = []
dic = {}
for i in featureList:
    dic[i[0]] = i[1]
for i in fcluster:
    if(dic[i[0]]=="NN"):
        finalcluster.append(i)
print(finalcluster)
```

```
[['soundquality', ['great']], ['batterylife', ['bad']]]
```