In [ ]:
```
#LDA
#Clean the data to remove stop-words, punctuation, and emoticons (similar to last week'
#Apply LDA and print out 10 topics
#Chatbot
#Clean the data as you did for LDA
#Build a chatbot based on the reviews
```

In [1]:
```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.metrics.pairwise import cosine_similarity
import string
import random
import nltk
import pandas as pd
import numpy as np

# Opening the file
f = open("C:/Users/benso/Desktop/Projects/Usable_AI_Code/Homework10/amazon_cells_labell

data =[]
# Converting it to pandas dataframe
for line in f:
    review = line[:len(line) - 2]
    sentiment = "neg" if line[len(line)-2] == "0" else "pos"
    row = [review, sentiment]
    data.append(row)

df = pd.DataFrame(data, columns = ['reviews', 'sentiment'])
```

In [2]:
```
#remove stopwords
from nltk.corpus import stopwords
stop = stopwords.words('english')

def stopwords(text):
    text = [word.lower() for word in text.split() if word.lower() not in stop]
    return " ".join(text)

df["reviews"] = df["reviews"].apply(stopwords)
df.head()
```

Out[2]:

	reviews	sentiment
0	way plug us unless go converter.	neg
1	good case, excellent value.	pos
2	great jawbone.	pos
3	tied charger conversations lasting 45 minutes....	neg
4	mic great.	pos

In [3]:
```
#punctuations
import string
df['reviews'].str.replace('[{}]'.format(string.punctuation), '')
```

```
ipykernel_launcher:3: FutureWarning: The default value of regex will change from True to
False in a future version.
```

Out[3]:
```
0                        way plug us unless go converter
1                              good case excellent value
2                                         great jawbone
3       tied charger conversations lasting 45 minutesm...
4                                             mic great
                             ...
995            screen get smudged easily touches ear face
996                          piece junk lose calls phone
997                                   item match picture
998                  thing disappoint infra red port irda
999                   answer calls unit never worked once
Name: reviews, Length: 1000, dtype: object
```

In [4]:
```python
#remove common words

from collections import Counter
cnt = Counter()
for text in df["reviews"].values:
    for word in text.split():
        cnt[word] += 1

cnt.most_common(10)

freq = set([w for (w, wc) in cnt.most_common(10)])
def freqwords(text):
    return " ".join([word for word in str(text).split() if word not
in freq])
df["reviews"] = df["reviews"].apply(freqwords)
df["reviews"].head()
```

Out[4]:
```
0                      way plug us unless go converter.
1                                 case, excellent value.
2                                              jawbone.
3     tied charger conversations lasting 45 minutes....
4                                            mic great.
Name: reviews, dtype: object
```

In [6]:
```python
#remove emoticons

import re
from emot.emo_unicode import UNICODE_EMO, EMOTICONS

def remove_emoticons(text):
    emoticon_pattern = re.compile(u'(' + u'|'.join(k for k in EMOTICONS) + u')')
    return emoticon_pattern.sub(r'', text)

df['reviews'] = df['reviews'].apply(remove_emoticons)
```

In [8]:
```python
#Apply LDA

#Lementize
"The goal of both stemming and lemmatization is to reduce inflectional forms and someti

from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
```

```
def lemmatizewords(text):
    #WordNet Lemmatization
    text = [WordNetLemmatizer().lemmatize(word) for word in text.split()]
    return " ".join(text)

df['reviews'] = df['reviews'].apply(lemmatizewords)
df.head(10)
```

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

Out[8]:

	reviews	sentiment
0	way plug u unless go converter.	neg
1	case, excellent value.	pos
2	jawbone.	pos
3	tied charger conversation lasting 45 minutes.m...	neg
4	mic great.	pos
5	jiggle plug get line right get decent volume.	neg
6	several dozen several hundred contacts, imagin...	neg
7	razr owner...you must this!	pos
8	needle say, wasted money.	neg
9	waste money time!.	neg

In [10]:

```
#LDA

import pandas as pd
import numpy as np

from sklearn.feature_extraction.text import CountVectorizer

vect = CountVectorizer(max_features=5000, max_df=.15)
X = vect.fit_transform(df['reviews'])

from sklearn.decomposition import LatentDirichletAllocation
lda = LatentDirichletAllocation(n_components=10, learning_method="batch", max_iter=25,

document_topics = lda.fit_transform(X)

print(lda.components_.shape)
document_topics
```

```
(10, 1711)
```

Out[10]:

```
array([[0.01666787, 0.01666667, 0.01666814, ..., 0.01667056, 0.01666682,
        0.84997746],
       [0.02500157, 0.02500082, 0.02500112, ..., 0.77498438, 0.02500108,
        0.02500543],
       [0.05      , 0.05      , 0.0500127 , ..., 0.05      , 0.05      ,
        0.5499873 ],
       ...,
       [0.02500359, 0.77497638, 0.02500118, ..., 0.02500257, 0.025     ,
        0.025     ],
       [0.01428571, 0.01428655, 0.01428606, ..., 0.01428597, 0.87142139,
        0.01429075],
       [0.01428667, 0.87142076, 0.01428624, ..., 0.01428626, 0.01428759,
        0.01428706]])
```

In [11]:

```
sorting = np.argsort(lda.components_, axis=1)[:, ::-1]
print(len(sorting))
print(sorting)
```

```
10
[[ 798 1095  421 ... 1063   23  296]
 [ 671 1220 1208 ...  153  296   23]
 [1101  810  482 ...  728  260 1331]
 ...
 [1156  526 1144 ...   23  153  296]
 [1594  702  482 ... 1063   23  153]
 [ 582  892 1689 ...  296   23   12]]
```

In [12]:

```
feature_names = np.array(vect.get_feature_names())
print(len(feature_names))
print(feature_names)
```

```
1711
['10' '100' '11' ... 'you' 'z500a' 'zero']
```

In [13]:

```
def print_topics(topics, feature_names, sorting, topics_per_chunk, n_words):
    for i in range(0, len(topics), topics_per_chunk):
        # for each chunk:
        these_topics = topics[i: i + topics_per_chunk]
        # maybe we have less than topics_per_chunk left
        len_this_chunk = len(these_topics)
        print(these_topics)
        print(*these_topics)
        print(len_this_chunk)
        # print topic headers
        print(("topic {:<8}" * len_this_chunk).format(*these_topics))
        print(("-------- {0:<5}" * len_this_chunk).format(""))
        # print top n_words frequent words
        for i in range(n_words):
            try:
                print(("{:<14}" * len_this_chunk).format(*feature_names[sorting[these_t
            except:
                pass
        print("\n")

print_topics(topics=range(10), feature_names=feature_names, sorting=sorting, topics_per
```

```
range(0, 5)
0 1 2 3 4
5
topic 0       topic 1       topic 2       topic 3       topic 4
--------      --------      --------      --------      --------
it            great         piece         time          could
phone         reception     junk          waste         use
disappointed  really        easy          like          it
samsung       charge        product       money         this
too           me            fine          ear           well
clear         call          it            volume        problem
better        also          best          product       like
ear           problem       button        work          take
```

```
price          worked         sound          audio          make
camera         horrible       ear            case           charger


range(5, 10)
5 6 7 8 9
5
topic 5        topic 6        topic 7        topic 8        topic 9
--------       --------       --------       --------       --------
recommend      service        product        use            fit
item           ever           excellent      headset        love
life           headset        price          easy           work
highly         worst          well           enough         comfortable
nice           customer       quality        looking        case
cool           best           great          ear            it
long           ve             happy          bluetooth      well
look           used           case           long           bluetooth
even           terrible       work           want           ear
ear            well           working        time           purchase
```

In [17]:

```python
#nltk.download('punkt') # first-time use only
#nltk.download('wordnet') # first-time use only
raw=f.read()

sent_tokens = nltk.sent_tokenize(raw)# converts to list of sentences
word_tokens = nltk.word_tokenize(raw)# converts to list of words
```

In [24]:

```python
#WordNet is a semantically-oriented dictionary of English included in NLTK.
lemmer = nltk.stem.WordNetLemmatizer()

def LemTokens(tokens):
    return [lemmer.lemmatize(token) for token in tokens]

remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)

# remove punctuation, tokenize, and lemmatize in one call
def LemNormalize(text):
    return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))
```

In [18]:

```python
#Chatbot

#already cleaned the data
#already lemmentized

# Default greeting messages
GREETING_INPUTS = ("hello", "hi", "greetings", "sup", "what's up","hey",)
GREETING_RESPONSES = ["hi", "hey", "*nods*", "hi there", "hello", "I am glad! You are t

def greeting(sentence):
    for word in sentence.split():
        # If user said hello, greet back
        if word.lower() in GREETING_INPUTS:
            return random.choice(GREETING_RESPONSES)
```

In [22]:

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

def response(user_response):
    robo_response=''

    # add user input to sentence tokens
    sent_tokens.append(user_response)
    # convert sentence tokens to TF-IDF feature matrix [document, word][idf]
    tfidfvec = TfidfVectorizer(tokenizer=LemNormalize, stop_words='english')
    tfidf = tfidfvec.fit_transform(sent_tokens)
    # calculate cosine similarity between user input to each TF-IDF document (sentence)
    vals = cosine_similarity(tfidf[-1], tfidf)
    # sort cosine similarity values in ascending order,
    # then select index of highest cosine similarity value, excluding user input
    idx=vals.argsort()[0][-2]
    # convert from 2D to 1D array
    flat = vals.flatten()
    # sort cosine similarity values in ascending order
    flat.sort()
    # selecting highest cosine similarity, exclusing user input
    similarity = flat[-2]
    if(similarity==0):
        robo_response=robo_response+"I am sorry! I don't understand you"
        return robo_response
    else:
        robo_response = robo_response+sent_tokens[idx]
        return robo_response
```

In [25]:

```
flag=True
print("ROBO: My name is Robo. I will answer your queries about Amazon reviews. If you w
while(flag==True):
    user_response = input()
    user_response=user_response.lower()
    if(user_response!='bye'):
        if(user_response=='thanks' or user_response=='thank you' ):
            flag=False
            print("ROBO: You are welcome..")
        else:
            if(greeting(user_response)!=None):
                print("ROBO: "+greeting(user_response))
            else:
                print("ROBO: ",end="")
                print(response(user_response))
                sent_tokens.remove(user_response)
    else:
        flag=False
        print("ROBO: Bye! take care..")
```

```
ROBO: My name is Robo. I will answer your queries about Amazon reviews. If you want to e
xit, type 'bye'!
C:\Users\benso\Anaconda3\lib\site-packages\sklearn\feature_extraction\text.py:391: UserW
arning: Your stop_words may be inconsistent with your preprocessing. Tokenizing the stop
words generated tokens ['ha', 'le', 'u', 'wa'] not in stop_words.
  'stop_words.' % sorted(inconsistent))
ROBO: how does amazon work?
ROBO: Bye! take care..
```

In [ ]: