360-Intel-Code

May 31, 2020

In [ ]:
```
# Importing libraries

import numpy as np
import pandas as pd
import os
from keras.preprocessing.text import text_to_word_sequence
from keras.preprocessing.text import one_hot
from keras.preprocessing.text import Tokenizer
from keras.utils.vis_utils import plot_model
from keras.models import Sequential
from keras.layers import Dense
import nltk
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk import wordpunct_tokenize
from nltk import sent_tokenize
from nltk import word_tokenize
from nltk import wordpunct_tokenize
from nltk import PunktSentenceTokenizer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
import string
import re
import re
from textblob import TextBlob
from textblob.sentiments import NaiveBayesAnalyzer
from sklearn.feature_extraction.text import CountVectorizer
```

In [ ]:
```
# Import directory

%cd C:/Users/Samantha/Documents/360 Data/fd_small

# ----------------------------------------------------------
# Pulling list of names of quickserve files in  folder
# ----------------------------------------------------------
finedining_files = []
```

```
        for f in os.listdir():
            finedining_files.append(f)
        print(len(finedining_files))
        sorted(finedining_files)

        # Get the list of files to change the files name for next step
```

In [ ]:
```
        # Import a file

        data_1 = pd.read_csv("C1_fd_aim_small.csv")
        print(data_1.shape)
        C1= data_1.filter(regex='com$',axis=1)
        print(C1.shape)
        C1.head()
```

In [ ]:
```
        # Extract only comment cols and check no. of each domain

        C1_G = C1.filter(regex='^G',axis=1)
        C1_C = C1.filter(regex='^C',axis=1)
        C1_S = C1.filter(regex='^S',axis=1)
        C1_T = C1.filter(regex='^T',axis=1)
        C1_Q = C1.filter(regex='^Q',axis=1)

        # Convert NAN to blank str
        C1_G = C1_G.fillna('')
        C1_C = C1_C.fillna('')
        C1_S = C1_S.fillna('')
        C1_T = C1_T.fillna('')
        C1_Q = C1_Q.fillna('')

        print("General : ",C1_G.shape)
        print("Cleanliness : ",C1_C.shape)
        print("Service : ",C1_S.shape)
        print("Timing : ",C1_T.shape)
        print("Quality : ",C1_Q.shape)
```

In [ ]:
```
        # ----------------------------------
        # Combining all reviews in one list
        # ----------------------------------


        # Then stack all sub-domain question to 1 whole list


        C1_S_list = C1_S.to_numpy().tolist()
        C1_Q_list = C1_Q.to_numpy().tolist()
        C1_G_list = C1_G.to_numpy().tolist()
        C1_T_list = C1_T.to_numpy().tolist()
```

```
C1_C_list = C1_C.to_numpy().tolist()


# Then convert to string

C1_S_str = str(C1_S_list)
C1_Q_str = str(C1_Q_list)
C1_G_str = str(C1_G_list)
C1_T_str = str(C1_T_list)
C1_C_str = str(C1_C_list)



# -----------------------------------
# Cleaning
# -----------------------------------


## split into words
NLTKtokens_C1_S = word_tokenize(C1_S_str)
NLTKtokens_C1_Q = word_tokenize(C1_Q_str)
NLTKtokens_C1_G = word_tokenize(C1_G_str)
NLTKtokens_C1_T = word_tokenize(C1_T_str)
NLTKtokens_C1_C = word_tokenize(C1_C_str)


## Remove Punctuation (anything that is not alphabetic)
NLTKwords_C1_S = [word for word in NLTKtokens_C1_S if word.isalpha()]
NLTKwords_C1_Q = [word for word in NLTKtokens_C1_Q if word.isalpha()]
NLTKwords_C1_G = [word for word in NLTKtokens_C1_G if word.isalpha()]
NLTKwords_C1_T = [word for word in NLTKtokens_C1_T if word.isalpha()]
NLTKwords_C1_C = [word for word in NLTKtokens_C1_C if word.isalpha()]


## convert to lower case
NLTKtokens_C1_S = [w.lower() for w in NLTKtokens_C1_S]
NLTKtokens_C1_Q = [w.lower() for w in NLTKtokens_C1_Q]
NLTKtokens_C1_G = [w.lower() for w in NLTKtokens_C1_G]
NLTKtokens_C1_T = [w.lower() for w in NLTKtokens_C1_T]
NLTKtokens_C1_C = [w.lower() for w in NLTKtokens_C1_C]



## prepare regex for char filtering
re_punc = re.compile('[%s]' % re.escape(string.punctuation))

## remove punctuation from each word
```

```
stripped_C1_S = [re_punc.sub('', w) for w in NLTKtokens_C1_S]
stripped_C1_Q = [re_punc.sub('', w) for w in NLTKtokens_C1_Q]
stripped_C1_G = [re_punc.sub('', w) for w in NLTKtokens_C1_G]
stripped_C1_T = [re_punc.sub('', w) for w in NLTKtokens_C1_T]
stripped_C1_C = [re_punc.sub('', w) for w in NLTKtokens_C1_C]



## remove remaining tokens that are not alphabetic
NLTKwords_C1_S = [word for word in stripped_C1_S if word.isalpha()]
NLTKwords_C1_Q = [word for word in stripped_C1_Q if word.isalpha()]
NLTKwords_C1_G = [word for word in stripped_C1_G if word.isalpha()]
NLTKwords_C1_T = [word for word in stripped_C1_T if word.isalpha()]
NLTKwords_C1_C = [word for word in stripped_C1_C if word.isalpha()]


## filter out stop words
stop_words = stopwords.words('english')
stop_words.extend([".","-","(", ")","/", ",", "", "","", "\n","nan"]) #Remove NAN

NLTKwords_C1_S = [w for w in NLTKwords_C1_S if not w in stop_words]
NLTKwords_C1_Q = [w for w in NLTKwords_C1_Q if not w in stop_words]
NLTKwords_C1_G = [w for w in NLTKwords_C1_G if not w in stop_words]
NLTKwords_C1_T = [w for w in NLTKwords_C1_T if not w in stop_words]
NLTKwords_C1_C = [w for w in NLTKwords_C1_C if not w in stop_words]


## stemming of words
# porter = PorterStemmer()

# NLTKstemmed_C1_S = [porter.stem(word) for word in NLTKwords_C1_S]
# NLTKstemmed_C1_Q = [porter.stem(word) for word in NLTKwords_C1_Q]
# NLTKstemmed_C1_G = [porter.stem(word) for word in NLTKwords_C1_G]
# NLTKstemmed_C1_T = [porter.stem(word) for word in NLTKwords_C1_T]
# NLTKstemmed_C1_C = [porter.stem(word) for word in NLTKwords_C1_C]


print("Customer 3, Service")
print(len(NLTKwords_C1_S))
print(len(set(NLTKwords_C1_S)))
print ("**************************************")
print("Customer 3, Quality")
print(len(NLTKwords_C1_Q))
print(len(set(NLTKwords_C1_Q)))
print ("**************************************")
print("Customer 3, General")
print(len(NLTKwords_C1_G))
print(len(set(NLTKwords_C1_G)))
```

```
print ("**************************************")
print("Customer 3, Timing")
print(len(NLTKwords_C1_T))
print(len(set(NLTKwords_C1_T)))
print ("**************************************")
print("Customer 3, Cleanliness")
print(len(NLTKwords_C1_C))
print(len(set(NLTKwords_C1_C)))
```

1 Extract words after applying polarity score

In [ ]:
```
# Storing polarity and subjectivity in lists using data from NLTKstemmed process

pol_C1_S = []
subj_C1_S = []

pol_C1_Q = []
subj_C1_Q = []

pol_C1_G = []
subj_C1_G = []

pol_C1_T = []
subj_C1_T = []

pol_C1_C = []
subj_C1_C = []

for i in NLTKwords_C1_S:
    test_C1_S = TextBlob(i)
    obj_C1_S = test_C1_S.sentiment
    pol_C1_S.append(obj_C1_S[0])
    subj_C1_S.append(obj_C1_S[1])

for i in NLTKwords_C1_Q:
    test_C1_Q = TextBlob(i)
    obj_C1_Q = test_C1_Q.sentiment
    pol_C1_Q.append(obj_C1_Q[0])
    subj_C1_Q.append(obj_C1_Q[1])

for i in NLTKwords_C1_G:
    test_C1_G = TextBlob(i)
    obj_C1_G = test_C1_G.sentiment
    pol_C1_G.append(obj_C1_G[0])
    subj_C1_G.append(obj_C1_G[1])

for i in NLTKwords_C1_T:
```

```
        test_C1_T = TextBlob(i)
        obj_C1_T = test_C1_T.sentiment
        pol_C1_T.append(obj_C1_T[0])
        subj_C1_T.append(obj_C1_T[1])

    for i in NLTKwords_C1_C:
        test_C1_C = TextBlob(i)
        obj_C1_C = test_C1_C.sentiment
        pol_C1_C.append(obj_C1_C[0])
        subj_C1_C.append(obj_C1_C[1])


    pola_score_C1_S = pd.DataFrame(pol_C1_S, columns = ['Polarity'])
    pola_score_C1_Q = pd.DataFrame(pol_C1_Q, columns = ['Polarity'])
    pola_score_C1_G = pd.DataFrame(pol_C1_G, columns = ['Polarity'])
    pola_score_C1_T = pd.DataFrame(pol_C1_T, columns = ['Polarity'])
    pola_score_C1_C = pd.DataFrame(pol_C1_C, columns = ['Polarity'])

    pola_score_C1_S.head()
```

In [ ]:
```
# Extract element & count a list of words


    # extract the word

    df_stem_C1_S = pd.DataFrame(NLTKwords_C1_S, columns =['word_C1_S'])
    df_stem_C1_Q = pd.DataFrame(NLTKwords_C1_Q, columns =['word_C1_Q'])
    df_stem_C1_G = pd.DataFrame(NLTKwords_C1_G, columns =['word_C1_G'])
    df_stem_C1_T = pd.DataFrame(NLTKwords_C1_T, columns =['word_C1_T'])
    df_stem_C1_C = pd.DataFrame(NLTKwords_C1_C, columns =['word_C1_C'])

    print(df_stem_C1_S.head(5))
```

In [ ]:
```
## Extract the frequency

    count_C1_S = []
    count_C1_Q = []
    count_C1_G = []
    count_C1_T = []
    count_C1_C = []

    for i in NLTKwords_C1_S:
        c_C1_S = NLTKwords_C1_S.count(i)
        count_C1_S.append(c_C1_S)

    for i in NLTKwords_C1_Q:
        c_C1_Q = NLTKwords_C1_Q.count(i)
        count_C1_Q.append(c_C1_Q)
```

```
for i in NLTKwords_C1_G:
    c_C1_G = NLTKwords_C1_G.count(i)
    count_C1_G.append(c_C1_G)

for i in NLTKwords_C1_T:
    c_C1_T = NLTKwords_C1_T.count(i)
    count_C1_T.append(c_C1_T)

for i in NLTKwords_C1_C:
    c_C1_C = NLTKwords_C1_C.count(i)
    count_C1_C.append(c_C1_C)

print(count_C1_S)
```

In [ ]:
```
### Put into DF

df_count_C1_S = pd.DataFrame(count_C1_S, columns = ['count'])
df_count_C1_Q = pd.DataFrame(count_C1_Q, columns = ['count'])
df_count_C1_G = pd.DataFrame(count_C1_G, columns = ['count'])
df_count_C1_T = pd.DataFrame(count_C1_T, columns = ['count'])
df_count_C1_C = pd.DataFrame(count_C1_C, columns = ['count'])



concat_C1_S = pd.concat([df_stem_C1_S,pola_score_C1_S['Polarity']],axis=1)
concat_C1_Q = pd.concat([df_stem_C1_Q,pola_score_C1_Q['Polarity']],axis=1)
concat_C1_G = pd.concat([df_stem_C1_G,pola_score_C1_G['Polarity']],axis=1)
concat_C1_T = pd.concat([df_stem_C1_T,pola_score_C1_T['Polarity']],axis=1)
concat_C1_C = pd.concat([df_stem_C1_C,pola_score_C1_C['Polarity']],axis=1)



### Extract only unique word

uni_concat_C1_S = concat_C1_S.word_C1_S.drop_duplicates()
uni_concat_C1_Q = concat_C1_Q.word_C1_Q.drop_duplicates()
uni_concat_C1_G = concat_C1_G.word_C1_G.drop_duplicates()
uni_concat_C1_T = concat_C1_T.word_C1_T.drop_duplicates()
uni_concat_C1_C = concat_C1_C.word_C1_C.drop_duplicates()


### Put into DF

df_uni_C1_S = uni_concat_C1_S.to_frame()
df_uni_C1_Q = uni_concat_C1_Q.to_frame()
df_uni_C1_G = uni_concat_C1_G.to_frame()
df_uni_C1_T = uni_concat_C1_T.to_frame()
```

```
df_uni_C1_C = uni_concat_C1_C.to_frame()
```

```
### Concate back to Polarity score

C1_S_drop = pd.concat([df_uni_C1_S,pola_score_C1_S['Polarity']],axis=1)
C1_Q_drop = pd.concat([df_uni_C1_Q,pola_score_C1_Q['Polarity']],axis=1)
C1_G_drop = pd.concat([df_uni_C1_G,pola_score_C1_G['Polarity']],axis=1)
C1_T_drop = pd.concat([df_uni_C1_T,pola_score_C1_T['Polarity']],axis=1)
C1_C_drop = pd.concat([df_uni_C1_C,pola_score_C1_C['Polarity']],axis=1)
```

```
### Drop Na after concatenate
C1_S_drop_dropNA = C1_S_drop.dropna()
C1_Q_drop_dropNA = C1_Q_drop.dropna()
C1_G_drop_dropNA = C1_G_drop.dropna()
C1_T_drop_dropNA = C1_T_drop.dropna()
C1_C_drop_dropNA = C1_C_drop.dropna()
```

```
### Final DF

final_C1_S = pd.concat([C1_S_drop_dropNA,df_count_C1_S], axis = 1)
final_C1_Q = pd.concat([C1_Q_drop_dropNA,df_count_C1_Q], axis = 1)
final_C1_G = pd.concat([C1_G_drop_dropNA,df_count_C1_G], axis = 1)
final_C1_T = pd.concat([C1_T_drop_dropNA,df_count_C1_T], axis = 1)
final_C1_C = pd.concat([C1_C_drop_dropNA,df_count_C1_C], axis = 1)
```

```
### Drop Na again after concatenate back with score

final_C1_S = final_C1_S.dropna()
final_C1_Q = final_C1_Q.dropna()
final_C1_G = final_C1_G.dropna()
final_C1_T = final_C1_T.dropna()
final_C1_C = final_C1_C.dropna()
```

```
final_C1_S.head()
```

In [ ]:
```
# Top POSITIVE

pos_C1_S = final_C1_S.sort_values(by='Polarity', ascending=False)
pos_C1_Q = final_C1_Q.sort_values(by='Polarity', ascending=False)
pos_C1_G = final_C1_G.sort_values(by='Polarity', ascending=False)
```

```
pos_C1_T = final_C1_T.sort_values(by='Polarity', ascending=False)
pos_C1_C = final_C1_C.sort_values(by='Polarity', ascending=False)


print(pos_C1_S.head(20))

# Top NEGATIVE

neg_C1_S = final_C1_S.sort_values(by='Polarity', ascending=True)
neg_C1_Q = final_C1_Q.sort_values(by='Polarity', ascending=True)
neg_C1_G = final_C1_G.sort_values(by='Polarity', ascending=True)
neg_C1_T = final_C1_T.sort_values(by='Polarity', ascending=True)
neg_C1_C = final_C1_C.sort_values(by='Polarity', ascending=True)


neg_C1_S.head(20)
```

In [ ]:
```
# Export files

# Positive

pos_C1_S.to_csv('pos_C1_S.csv', index = False)
pos_C1_Q.to_csv('pos_C1_Q.csv', index = False)
pos_C1_G.to_csv('pos_C1_G.csv', index = False)
pos_C1_T.to_csv('pos_C1_T.csv', index = False)
pos_C1_C.to_csv('pos_C1_C.csv', index = False)

# Negative

neg_C1_S.to_csv('neg_C1_S.csv', index = False)
neg_C1_Q.to_csv('neg_C1_Q.csv', index = False)
neg_C1_G.to_csv('neg_C1_G.csv', index = False)
neg_C1_T.to_csv('neg_C1_T.csv', index = False)
neg_C1_C.to_csv('neg_C1_C.csv', index = False)
```

In [ ]: