In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import seaborn as sns
import nltk
import re
import re,string,unicodedata
from nltk.corpus import stopwords

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential

from unidecode import unidecode
import collections
from wordcloud import WordCloud

from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer, WordNetLemmatizer

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.metrics import accuracy_score, classification_report

import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
import pandas as pd
df = pd.read_json("Sarcasm_Headlines_Dataset.json", lines=True)
df.head()
```

Out[2]:

	article_link	headline	is_sarcastic
0	https://www.huffingtonpost.com/entry/versace-b...	former versace store clerk sues over secret 'b...	0
1	https://www.huffingtonpost.com/entry/roseanne-...	the 'roseanne' revival catches up to our thorn...	0
2	https://local.theonion.com/mom-starting-to-fea...	mom starting to fear son's web series closest ...	1
3	https://politics.theonion.com/boehner-just-wan...	boehner just wants wife to listen, not come up...	1
4	https://www.huffingtonpost.com/entry/jk-rowlin...	j.k. rowling wishes snape happy birthday in th...	0

In [3]:

```
df.shape
```

Out[3]:

```
(26709, 3)
```

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26709 entries, 0 to 26708
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   article_link  26709 non-null  object
 1   headline      26709 non-null  object
 2   is_sarcastic  26709 non-null  int64 
dtypes: int64(1), object(2)
memory usage: 626.1+ KB
```

In [5]:

```
df.isnull().sum()
```

Out[5]:

```
article_link    0
headline        0
is_sarcastic    0
dtype: int64
```

In [6]:

```
df.drop('article_link',axis = 1)
```

Out[6]:

	headline	is_sarcastic
0	former versace store clerk sues over secret 'b...	0
1	the 'roseanne' revival catches up to our thorn...	0
2	mom starting to fear son's web series closest ...	1
3	boehner just wants wife to listen, not come up...	1
4	j.k. rowling wishes snape happy birthday in th...	0
...	...	...
26704	american politics in moral free-fall	0
26705	america's best 20 hikes	0
26706	reparations and obama	0
26707	israeli ban targeting boycott supporters raise...	0
26708	gourmet gifts for the foodie 2014	0

26709 rows × 2 columns

In [16]:

```
# taken from a refernce, Don't know the original author of the code.
import nltk
nltk.download('stopwords')
stop = set(stopwords.words('english'))
punctuation = list(string.punctuation)
stop.update(punctuation)
```

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

In [17]:

```
#Removing the stopwords from text
def split_into_words(text):
    # split into words by white space
    words = text.split()
    return words
```

In [18]:

```
def to_lower_case(words):
    # convert to lower case
    words = [word.lower() for word in words]
    return words
```

In [19]:

```
def remove_punctuation(words):
    # prepare regex for char filtering
    re_punc = re.compile('[%s]' % re.escape(string.punctuation))
    # remove punctuation from each word
    stripped = [re_punc.sub('', w) for w in words]
    return stripped
```

In [20]:

```
def keep_alphabetic(words):
    # remove remaining tokens that are not alphabetic
    words = [word for word in words if word.isalpha()]
    return words
```

In [21]:

```
def remove_stopwords(words):
    # filter out stop words
    stop_words = set(stopwords.words('english'))
    words = [w for w in words if not w in stop_words]
    return words
```

In [22]:

```


def to_sentence(words):
    # join words to a sentence
    return ' '.join(words)
```

In [23]:

```
#Removing the noisy text
def denoise_text(text):
    words = split_into_words(text)
    words = to_lower_case(words)
    words = remove_punctuation(words)
    words = keep_alphabetic(words)
    words = remove_stopwords(words)
    return to_sentence(words)
```

In [ ]:

```

```

In [24]:

```
#Apply function on review column
df['headline']=df['headline'].apply(denoise_text)
```

In [25]:

```
labels = (df['is_sarcastic'])
data = (df['headline'])
```

In [26]:

```
train_ratio = 0.80

train_size = int(len(labels)*train_ratio)

train_data = data[:train_size]
train_labels= labels[:train_size]

test_data = data[train_size:]
test_labels = labels[train_size:]
```

In [27]:

```
tokenizer = Tokenizer(oov_token='<OOV>')
tokenizer.fit_on_texts(train_data)

vocab_size = len(tokenizer.word_index)
print(vocab_size)

train_sequences = tokenizer.texts_to_sequences(train_data)
test_sequences = tokenizer.texts_to_sequences(test_data)
```

```
24524
```

In [28]:

```
maxlen=max([len(i) for i in train_sequences])
```

In [29]:

```
train_padded = pad_sequences(train_sequences, maxlen=maxlen,  padding='post')
test_padded = pad_sequences(test_sequences, maxlen=maxlen,  padding='post')
```

In [30]:

```
# Print a sample headline
index = 10
print(f'sample headline: {train_sequences[index]}')
print(f'padded sequence: {train_padded[index]} \n')

print(f'Original Sentence:  \n {tokenizer.sequences_to_texts(train_sequences[index:index

# Print dimensions of padded sequences
print(f'shape of padded sequences: {train_padded.shape}')
```

```
sample headline: [3058, 1796, 4194, 4, 4774, 6814, 1797, 827]
padded sequence: [3058 1796 4194    4 4774 6814 1797  827    0    0    0
0    0    0
    0    0    0    0    0    0    0    0    0    0    0]

Original Sentence:
 ['airline passengers tackle man rushes cockpit bomb threat']

shape of padded sequences: (21367, 25)
```

In [31]:

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size+1,100,input_length=maxlen),
    tf.keras.layers.Bidirectional( tf.keras.layers.LSTM(128)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.50),
    tf.keras.layers.Dense(64,activation='relu'),
    tf.keras.layers.Dense(1,activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_1 (Embedding)     (None, 25, 100)           2452500   
                                                                 
 bidirectional_1 (Bidirecti  (None, 256)               234496    
 onal)                                                           
                                                                 
 flatten_1 (Flatten)         (None, 256)               0         
                                                                 
 dropout_1 (Dropout)         (None, 256)               0         
                                                                 
 dense_2 (Dense)             (None, 64)                16448     
                                                                 
 dense_3 (Dense)             (None, 1)                 65        
                                                                 
=================================================================
Total params: 2703509 (10.31 MB)
Trainable params: 2703509 (10.31 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [32]:

```
history=model.fit(train_padded, np.array(train_labels),validation_data = (test_padded,np
```

```
Epoch 1/5
668/668 - 99s - loss: 0.4840 - accuracy: 0.7588 - val_loss: 0.4110 - val_a
ccuracy: 0.8117 - 99s/epoch - 148ms/step
Epoch 2/5
668/668 - 90s - loss: 0.2150 - accuracy: 0.9142 - val_loss: 0.4735 - val_a
ccuracy: 0.8021 - 90s/epoch - 135ms/step
Epoch 3/5
668/668 - 88s - loss: 0.0886 - accuracy: 0.9675 - val_loss: 0.6540 - val_a
ccuracy: 0.7885 - 88s/epoch - 132ms/step
Epoch 4/5
668/668 - 87s - loss: 0.0413 - accuracy: 0.9855 - val_loss: 0.8064 - val_a
ccuracy: 0.7849 - 87s/epoch - 131ms/step
Epoch 5/5
668/668 - 92s - loss: 0.0259 - accuracy: 0.9915 - val_loss: 1.1646 - val_a
ccuracy: 0.7855 - 92s/epoch - 137ms/step
```

In [ ]: