

```
In [16]: 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

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

```
In [17]: import pandas as pd
df = pd.read_json("Sarcasm_Headlines_Dataset_v2.json", lines=True)
df.head()
```

```
Out[17]:
```

	is_sarcastic	headline	article_link
0	1	thirtysomething scientists unveil doomsday clo...	https://www.theonion.com/thirtysomething-scienc...
1	0	dem rep. totally nails why congress is falling...	https://www.huffingtonpost.com/entry/donna-edw...
2	0	eat your veggies: 9 deliciously different recipes	https://www.huffingtonpost.com/entry/eat-your-...
3	1	inclement weather prevents liar from getting t...	https://local.theonion.com/inclement-weather-p...
4	1	mother comes pretty close to using word 'strea...	https://www.theonion.com/mother-comes-pretty-c...

```
In [18]: df.head()
```

```
Out[18]:
```

	is_sarcastic	headline	article_link
0	1	thirtysomething scientists unveil doomsday clo...	https://www.theonion.com/thirtysomething-scienc...
1	0	dem rep. totally nails why congress is falling...	https://www.huffingtonpost.com/entry/donna-edw...
2	0	eat your veggies: 9 deliciously different recipes	https://www.huffingtonpost.com/entry/eat-your-...
3	1	inclement weather prevents liar from getting t...	https://local.theonion.com/inclement-weather-p...
4	1	mother comes pretty close to using word 'strea...	https://www.theonion.com/mother-comes-pretty-c...

```
In [19]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28619 entries, 0 to 28618
Data columns (total 3 columns):
#   Column          Non-Null Count  Dtype
---  -
0   is_sarcastic    28619 non-null  int64
1   headline        28619 non-null  object
2   article_link    28619 non-null  object
dtypes: int64(1), object(2)
memory usage: 670.9+ KB
```

```
In [20]: df.shape
```

```
Out[20]: (28619, 3)
```

```
In [21]: #checking for null values in train data
df.isnull().sum()
```

```
Out[21]: is_sarcastic    0
headline        0
article_link     0
dtype: int64
```

```
In [22]: df.describe(include='object')
```

```
Out[22]:
```

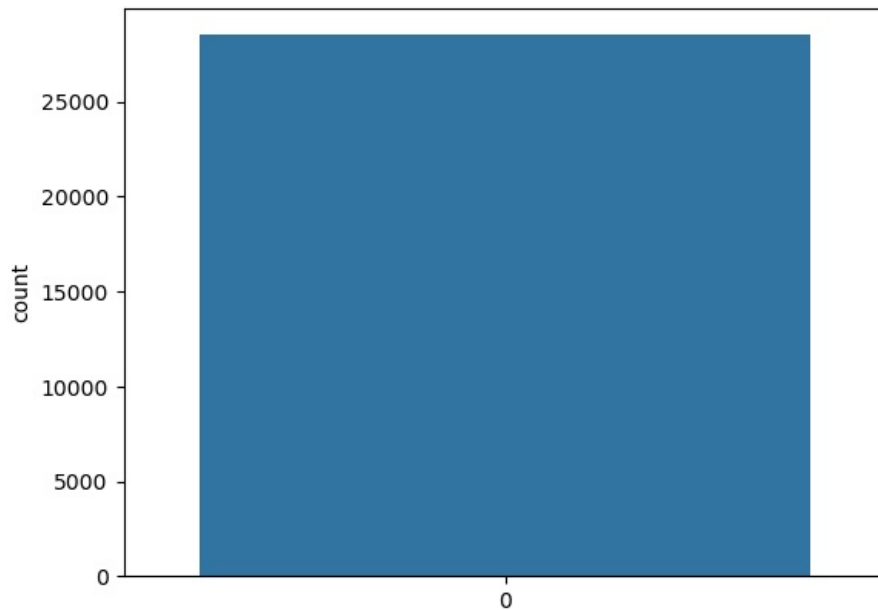
	headline	article_link
count	28619	28619
unique	28503	28617
top	'no way to prevent this,' says only nation whe...	https://politics.theonion.com/nation-not-sure-...
freq	12	2

```
In [23]: #checking for duplicate values
df['headline'].duplicated().sum()
```

```
Out[23]: 116
```

```
In [24]: df=df.drop(df[df['headline'].duplicated()].index,axis=0)
```

```
In [25]: sns.countplot(df['is_sarcastic']);
```



```
In [26]: # this function is copied from another kernel. Don't know who is the original author of it.
```

```
import nltk
nltk.download('stopwords')
stop = set(stopwords.words('english'))
punctuation = list(string.punctuation)
stop.update(punctuation)

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

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

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

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

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

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

#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)
```

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

```
In [27]: #Apply function on review column
df['headline']=df['headline'].apply(denoise_text)
```

```
In [28]: labels = (df['is_sarcastic'])
```

```
labels = ['1', '0']
data = (df['headline'])
```

```
In [29]: 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 [30]: 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)

25662
```

```
In [31]: maxlen = max([len(i) for i in train_sequences])
```

```
In [32]: train_padded = pad_sequences(train_sequences, maxlen=maxlen, padding='post')
test_padded = pad_sequences(test_sequences, maxlen=maxlen, padding='post')
```

```
In [33]: # 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+1])} \n')

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

```
sample headline: [1972, 2572, 315, 3022, 943, 7]
padded sequence: [1972 2572 315 3022 943 7 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0]
```

Original Sentence:
['lesbian considered father indiana amazing one']

shape of padded sequences: (22802, 106)

```
In [34]: 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"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 106, 100)	2566300
bidirectional (Bidirectional)	(None, 256)	234496
flatten (Flatten)	(None, 256)	0
dropout (Dropout)	(None, 256)	0
dense (Dense)	(None, 64)	16448
dense_1 (Dense)	(None, 1)	65

=====
Total params: 2817309 (10.75 MB)
Trainable params: 2817309 (10.75 MB)
Non-trainable params: 0 (0.00 Byte)

```
In [35]: history=model.fit(train_padded, np.array(train_labels),validation_data = (test_padded,np.array(test_labels)) ,
```

```
Epoch 1/5
713/713 - 262s - loss: 0.4810 - accuracy: 0.7566 - val_loss: 0.4027 - val_accuracy: 0.8132 - 262s/epoch - 367ms
/step
Epoch 2/5
713/713 - 255s - loss: 0.2079 - accuracy: 0.9187 - val_loss: 0.4597 - val_accuracy: 0.8081 - 255s/epoch - 357ms
/step
Epoch 3/5
713/713 - 323s - loss: 0.0899 - accuracy: 0.9668 - val_loss: 0.6117 - val_accuracy: 0.7974 - 323s/epoch - 452ms
/step
Epoch 4/5
713/713 - 329s - loss: 0.0665 - accuracy: 0.9827 - val_loss: 0.5877 - val_accuracy: 0.7115 - 329s/epoch - 461ms
/step
Epoch 5/5
713/713 - 307s - loss: 0.0448 - accuracy: 0.9847 - val_loss: 0.9116 - val_accuracy: 0.7907 - 307s/epoch - 430ms
/step
```

```
In [36]: import matplotlib.pyplot as plt
```

```
# Plot utility
def plot_graphs(model, string):
    plt.plot(model.history[string])
    plt.plot(model.history['val_'+string])
    plt.xlabel("Epochs")
    plt.ylabel(string)
    plt.legend([string, 'val_'+string])
    plt.show()

# Plot the accuracy and loss
plot_graphs(history, "accuracy")
plot_graphs(history, "loss")
```

