1. Design a NLP model on Sarcasm detection.

Electronic journalism powered with Social media has become one of the major sources of information consumption lately. Many media houses are using creative ways in order to tap into increasing views on posts. One of the ways is using sarcastic headlines as click baits. A model that is able to predict whether a piece of headline is sarcastic or not can be useful for media houses in order to analyse their quarterly earnings by strategy. Also, from a reader's perspective, search engines can utilise this information of sarcasm and depending on the reader's preference, recommend similar articles to them.

The goal is to build a ANN model to detect whether a sentence is sarcastic or not?

https://github.com/Kavitha-Kothandaraman/Sarcasm-Detection-NLP

In [1]:
```
import tensorflow as tf

# Display the version
print(tf.__version__)

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import matplotlib.pyplot as plt # plotting library
%matplotlib inline

from keras.models import Sequential
from keras.layers import Dense , Activation, Dropout
from keras.optimizers import Adam ,RMSprop
from keras import  backend as K
```

```
2.13.0
```

In [2]:
```
import pandas as pd
import os

data = pd.read_json(os.path.join('Sarcasm_Headlines_Dataset.json'),lines=True)
```

In [3]:
```
data
```

Out[3]:

| | 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 |
| **...** | ... | ... | ... |
| **26704** | https://www.huffingtonpost.com/entry/american-... | american politics in moral free-fall | 0 |
| **26705** | https://www.huffingtonpost.com/entry/americas-... | america's best 20 hikes | 0 |
| **26706** | https://www.huffingtonpost.com/entry/reparatio... | reparations and obama | 0 |
| **26707** | https://www.huffingtonpost.com/entry/israeli-b... | israeli ban targeting boycott supporters raise... | 0 |
| **26708** | https://www.huffingtonpost.com/entry/gourmet-g... | gourmet gifts for the foodie 2014 | 0 |

26709 rows × 3 columns

In [4]:
```
print (data.shape)
data.describe()
```

```
(26709, 3)
```

Out[4]:

| | is_sarcastic |
|---|---|
| **count** | 26709.000000 |
| **mean** | 0.438953 |
| **std** | 0.496269 |
| **min** | 0.000000 |
| **25%** | 0.000000 |
| **50%** | 0.000000 |
| **75%** | 1.000000 |
| **max** | 1.000000 |

In [5]:
```
data['headline'][1]
```

Out[5]:
```
"the 'roseanne' revival catches up to our thorny political mood, for better and worse"
```

In [6]:
```
##The column headline needs to be cleaned up as we have special characters and numbers in the column

import re
from nltk.corpus import stopwords
import nltk
import string
nltk.download('stopwords')
stopwords = set(stopwords.words('english'))
def cleanData(text):
  text = re.sub(r'\d+', '', text)
  text = "".join([char for char in text if char not in string.punctuation])
  return text

data['headline']=data['headline'].apply(cleanData)
```

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

In [7]:
```
data['headline'][1]
```

Out[7]:
```
'the roseanne revival catches up to our thorny political mood for better and worse'
```

In [8]:
```
data.drop('article_link',inplace=True,axis=1)
```

In [9]:
```
maxlen = max([len(text) for text in data['headline']])
print(maxlen)
```

```
240
```

In [10]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Dense, Input, LSTM, Embedding, Dropout, Activation, Flatten, Bidirectional,
from tensorflow.keras.models import Model, Sequential
```

In [11]:
```
tokenizer = Tokenizer(num_words=10000,filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',lower=True,split=' ', char
tokenizer.fit_on_texts(data['headline'])
```

In [12]:
```
num_words=len(tokenizer.word_index)

print (num_words)
```

```
27667
```

In [13]:
```
sentences = data['headline'].tolist()
labels = data['is_sarcastic'].tolist()

# Separate out the sentences and labels into training and test sets
training_size = int(len(sentences) * 0.8)

training_sentences = sentences[0:training_size]
testing_sentences = sentences[training_size:]

training_labels = labels[0:training_size]
testing_labels = labels[training_size:]

# Make labels into numpy arrays for use with the network later
training_labels_final = np.array(training_labels)
testing_labels_final = np.array(testing_labels)
```

In [14]:
```
vocab_size = 10000
embedding_dim = 16
max_length = 10000
trunc_type='post'
padding_type='post'
oov_tok = "<OOV>"


from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(training_sentences)
padded = pad_sequences(sequences,maxlen=max_length, padding=padding_type,
                       truncating=trunc_type)

testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded = pad_sequences(testing_sequences,maxlen=max_length,
                               padding=padding_type, truncating=trunc_type)
```

In [15]:
```
padded
```

Out[15]:
```
array([[ 300,    1,  805, ...,    0,    0,    0],
       [   4, 6946, 2914, ...,    0,    0,    0],
       [ 148,  898,    2, ...,    0,    0,    0],
       ...,
       [ 952, 3507,    5, ...,    0,    0,    0],
       [3032,    1,   12, ...,    0,    0,    0],
       [1154,  983,  209, ...,    0,    0,    0]])
```

In [16]:
```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 10000, 16)         160000

 flatten (Flatten)           (None, 160000)            0

 dense (Dense)               (None, 6)                 960006

 dense_1 (Dense)             (None, 1)                 7

=================================================================
Total params: 1120013 (4.27 MB)
Trainable params: 1120013 (4.27 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [17]:
```
num_epochs = 10 #Confined Epochs to 10 as my system is having 4GB RAM and isn't able to execute for higher Epoc
history=model.fit(padded, training_labels_final, epochs=num_epochs, validation_data=(testing_padded, testing_la
```

```
Epoch 1/10
668/668 [==============================] - 186s 172ms/step - loss: 0.6886 - accuracy: 0.5560 - val_loss: 0.6846
- val_accuracy: 0.5680
Epoch 2/10
668/668 [==============================] - 58s 86ms/step - loss: 0.6862 - accuracy: 0.5593 - val_loss: 0.6842 -
val_accuracy: 0.5680
Epoch 3/10
668/668 [==============================] - 58s 87ms/step - loss: 0.6861 - accuracy: 0.5593 - val_loss: 0.6840 -
val_accuracy: 0.5680
Epoch 4/10
668/668 [==============================] - 60s 90ms/step - loss: 0.6861 - accuracy: 0.5593 - val_loss: 0.6841 -
val_accuracy: 0.5680
Epoch 5/10
668/668 [==============================] - 60s 89ms/step - loss: 0.6861 - accuracy: 0.5593 - val_loss: 0.6841 -
val_accuracy: 0.5680
Epoch 6/10
668/668 [==============================] - 59s 88ms/step - loss: 0.6861 - accuracy: 0.5593 - val_loss: 0.6840 -
val_accuracy: 0.5680
Epoch 7/10
668/668 [==============================] - 59s 88ms/step - loss: 0.6861 - accuracy: 0.5593 - val_loss: 0.6841 -
val_accuracy: 0.5680
Epoch 8/10
668/668 [==============================] - 59s 88ms/step - loss: 0.6861 - accuracy: 0.5593 - val_loss: 0.6840 -
val_accuracy: 0.5680
Epoch 9/10
668/668 [==============================] - 62s 92ms/step - loss: 0.6861 - accuracy: 0.5593 - val_loss: 0.6840 -
val_accuracy: 0.5680
Epoch 10/10
668/668 [==============================] - 60s 90ms/step - loss: 0.6861 - accuracy: 0.5593 - val_loss: 0.6840 -
val_accuracy: 0.5680
```

In [18]:
```
# First get the weights of the embedding layer

e = model.layers[0]
weights = e.get_weights()[0]
print(weights.shape) # shape: (vocab_size, embedding_dim)
```

```
(10000, 16)
```

In [19]:
```
# Use the model to predict sarcasm
sarcasm = ['Not my choice', 'How can this be true',
                'Everything was cold',
                'Everything was hot exactly as I wanted',
                'Everything was green',
                'the host seated us immediately',
                'they gave us free chocolate cake',
                'not sure about the wilted flowers on the table',
                'only works when I stand on tippy toes',
                'does not work when I stand on my head']
```

```
print(sarcasm)

# Create the sequences
padding_type='post'
sample_sequences = tokenizer.texts_to_sequences(sarcasm)
sarcasm_padded = pad_sequences(sample_sequences, padding=padding_type, maxlen=max_length)

classes = model.predict(sarcasm_padded)

# The closer the class is to 1, the more positive the review is deemed to be
for x in range(len(sarcasm)):
  print(sarcasm[x])
  print(classes[x])
  print('\n')
```

```
['Not my choice', 'How can this be true', 'Everything was cold', 'Everything was hot exactly as I wanted', 'Eve
rything was green', 'the host seated us immediately', 'they gave us free chocolate cake', 'not sure about the w
ilted flowers on the table', 'only works when I stand on tippy toes', 'does not work when I stand on my head']
1/1 [==============================] - 5s 5s/step
Not my choice
[0.43802047]


How can this be true
[0.43802047]


Everything was cold
[0.43802047]


Everything was hot exactly as I wanted
[0.43802047]


Everything was green
[0.43802047]


the host seated us immediately
[0.43802047]


they gave us free chocolate cake
[0.43802047]


not sure about the wilted flowers on the table
[0.43802047]


only works when I stand on tippy toes
[0.43802047]


does not work when I stand on my head
[0.43802047]
```