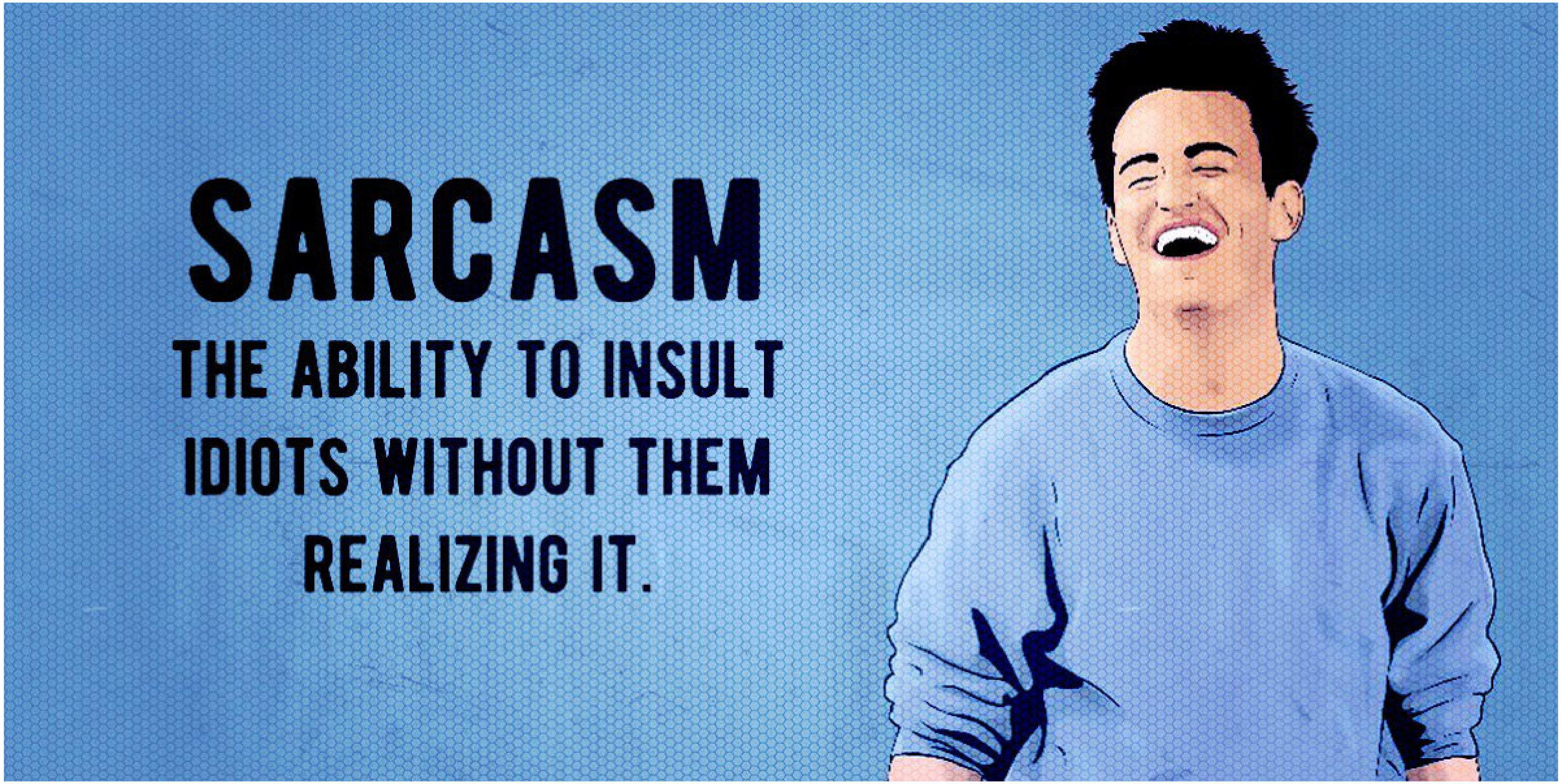


In [1]:
```
import json
import numpy as np
import pandas as pd
import tensorflow as tf
```

In [2]:
```
import wget
url='https://storage.googleapis.com/laurencemoroney-blog.appspot.com/sarcasm.json'
filename = wget.download(url)
filename
```

```
100% [.........................................] 5643545 / 5643545
```

Out[2]:
```
'sarcasm (1).json'
```

In [3]:

```
data=pd.read_json(filename)
data.head()
```

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

In [4]:

```
data.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 = data.iloc[:,1:]
df.head()
```

Out[5]:

	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

```
df.shape
```

In [6]:

```
```

Out[6]:

```
(26709, 2)
```

In [7]:

```
sentence = df.headline
labels = df.is_sarcastic
```

In [8]:

```
from sklearn.model_selection import train_test_split
X_train, X_test,y_train,y_test = train_test_split(sentence, labels, train_size=0.80,random_state=2)
```

In [9]:

```
X_train.shape, X_test.shape,y_train.shape,y_test.shape
```

Out[9]:

```
((21367,), (5342,), (21367,), (5342,))
```

In [10]:

```
from tensorflow.keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer(num_words=10000, oov_token="<OOV>")
tokenizer.fit_on_texts(X_train)
```

In [11]:

```
word_index = tokenizer.word_index
len(word_index)
```

Out[11]:

```
26527
```

In [12]:

```
training_sequences = tokenizer.texts_to_sequences(X_train)
training_sequences[0]
```

Out[12]:

```
[42, 103, 87, 69, 158, 94, 167, 170, 1, 23, 342, 4452]
```

In [13]:

```
from tensorflow.keras.preprocessing.sequence import pad_sequences

training_padded = pad_sequences(training_sequences,
                                maxlen=100,
                                padding='post',
                                truncating='post')
training_padded[0]
```

```
Out[13]: array([  42,  103,   87,   69,  158,   94,  167,  170,    1,   23,  342,
                4452,    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])
```

```
In [14]: testing_sequences = tokenizer.texts_to_sequences(X_test)

         testing_padded = pad_sequences(testing_sequences,
                                        maxlen=100,
                                        padding='post',
                                        truncating='post')
```

```
In [15]: testing_padded[0]
```

```
Out[15]: array([2496,   30,    4,   12,   27,    1,    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])
```

```
In [16]: #Layer Normalization
         model = tf.keras.Sequential([
             tf.keras.layers.Embedding(10000, 16, input_length=100),
             tf.keras.layers.GlobalAveragePooling1D(),
             tf.keras.layers.Dense(24, activation='relu'),
             tf.keras.layers.Dense(1, activation='sigmoid')
         ])

         model.compile(optimizer='adam',
                       loss='binary_crossentropy',
                       metrics=['accuracy'])
```

In [17]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, 100, 16)           160000    
                                                                 
 global_average_pooling1d (G  (None, 16)               0         
 lobalAveragePooling1D)                                          
                                                                 
 dense (Dense)               (None, 24)                408       
                                                                 
 dense_1 (Dense)             (None, 1)                 25        
                                                                 
=================================================================
Total params: 160,433
Trainable params: 160,433
Non-trainable params: 0
_________________________________________________________________
```

In [18]:
```
history = model.fit(training_padded,
                    y_train,
                    epochs=10,
                    validation_data=(testing_padded, y_test),
                    verbose=2)
```

```
Epoch 1/10
668/668 - 3s - loss: 0.6633 - accuracy: 0.5910 - val_loss: 0.5654 - val_accuracy: 0.8162 - 3s/epoch - 5ms/step
Epoch 2/10
668/668 - 3s - loss: 0.4194 - accuracy: 0.8327 - val_loss: 0.3668 - val_accuracy: 0.8454 - 3s/epoch - 5ms/step
Epoch 3/10
668/668 - 3s - loss: 0.3069 - accuracy: 0.8769 - val_loss: 0.3385 - val_accuracy: 0.8519 - 3s/epoch - 5ms/step
Epoch 4/10
668/668 - 3s - loss: 0.2591 - accuracy: 0.8968 - val_loss: 0.3226 - val_accuracy: 0.8624 - 3s/epoch - 4ms/step
Epoch 5/10
668/668 - 3s - loss: 0.2255 - accuracy: 0.9119 - val_loss: 0.3204 - val_accuracy: 0.8663 - 3s/epoch - 4ms/step
Epoch 6/10
668/668 - 3s - loss: 0.2014 - accuracy: 0.9236 - val_loss: 0.3246 - val_accuracy: 0.8663 - 3s/epoch - 5ms/step
Epoch 7/10
668/668 - 3s - loss: 0.1806 - accuracy: 0.9328 - val_loss: 0.3326 - val_accuracy: 0.8658 - 3s/epoch - 5ms/step
Epoch 8/10
668/668 - 3s - loss: 0.1631 - accuracy: 0.9400 - val_loss: 0.3419 - val_accuracy: 0.8632 - 3s/epoch - 5ms/step
Epoch 9/10
668/668 - 3s - loss: 0.1488 - accuracy: 0.9463 - val_loss: 0.3758 - val_accuracy: 0.8553 - 3s/epoch - 5ms/step
Epoch 10/10
668/668 - 3s - loss: 0.1367 - accuracy: 0.9500 - val_loss: 0.3696 - val_accuracy: 0.8618 - 3s/epoch - 5ms/step
```

In [19]:

```
pd.DataFrame(history.history).head()
```

Out[19]:

	loss	accuracy	val_loss	val_accuracy
0	0.663316	0.591005	0.565383	0.816174
1	0.419370	0.832686	0.366813	0.845376
2	0.306855	0.876913	0.338508	0.851928
3	0.259105	0.896757	0.322640	0.862411
4	0.225541	0.911873	0.320427	0.866342

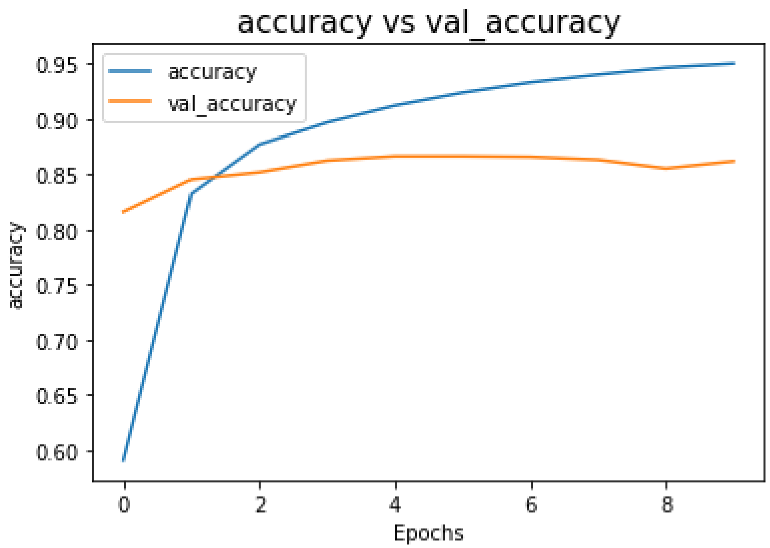
Plot accuracy vs val_accuracy & loss vs val_loss curve

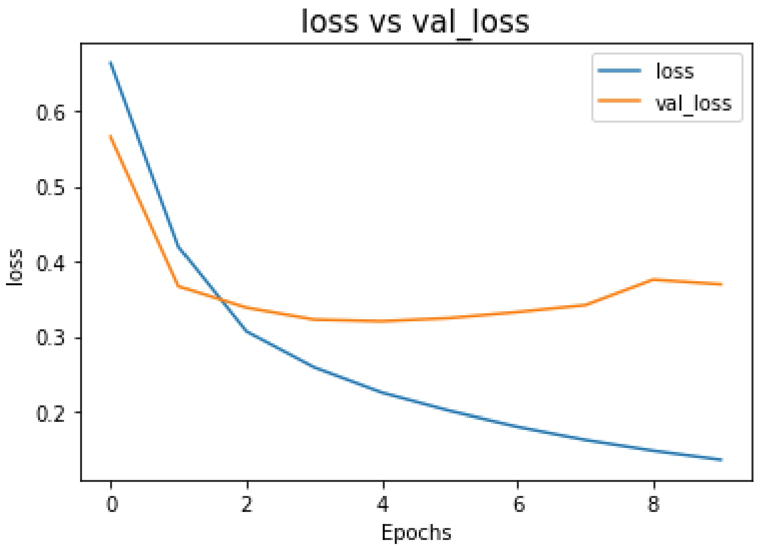
In [20]:

```
import matplotlib.pyplot as plt

def plot_graphs(history, string):
    plt.plot(history.history[string])
    plt.plot(history.history['val_'+string])
    plt.xlabel("Epochs")
    plt.ylabel(string)
    plt.legend([string, 'val_'+string])
    plt.title(f"{string} vs {'val_'+string}", fontsize = 15)
    plt.show()

plot_graphs(history, "accuracy")
plot_graphs(history, "loss")
```





In [21]:

```
model.evaluate(testing_padded,y_test)
```

```
167/167 [==============================] - 0s 1ms/step - loss: 0.3696 - accuracy: 0.8618
```

Out[21]:

```
[0.3695674240589142, 0.8618494868278503]
```

Predictions

In [22]:

```
pred = model.predict(testing_padded)
pred[:5]
```

Out[22]:

```
array([[0.20597145],
       [0.8800808 ],
       [0.35702163],
       [0.1383534 ],
       [0.99725986]], dtype=float32)
```

In [23]:

```
pred_labels= [1 * (i[0]>=0.5) for i in pred]
pred_labels[:5]
```

Out[23]:

```
[0, 1, 0, 0, 1]
```

In [24]:

```
y_test.head(5)
```

Out[24]:

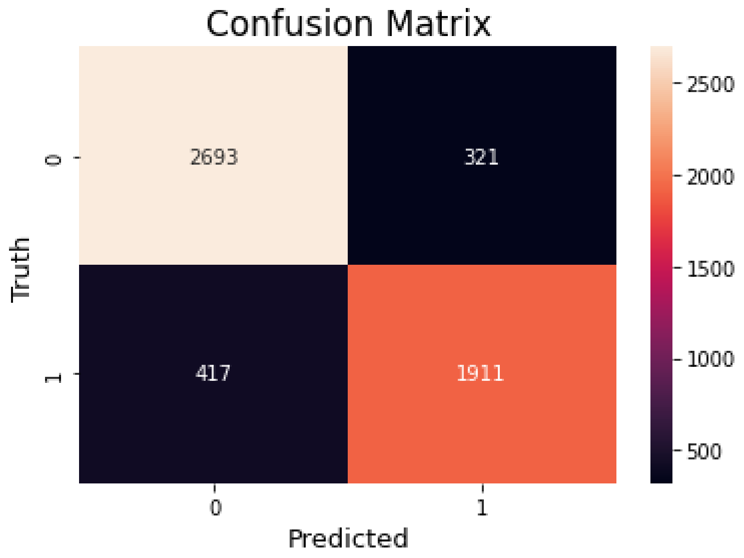
```
10465    0
970      1
17936    1
9001     0
12922    1
Name: is_sarcastic, dtype: int64
```

In [25]:

```
#Comfusion_Matrix = pd.crosstab(y_test, pred_labels)
Comfusion_Matrix=tf.math.confusion_matrix(labels = y_test, predictions= pred_labels)
```

In [26]:

```
import seaborn as sns
sns.heatmap(Comfusion_Matrix, annot = True, fmt='d')
plt.xlabel('Predicted', fontsize=13)
plt.ylabel('Truth', fontsize=13)
plt.title('Confusion Matrix', fontsize=17)
plt.show()
```



In [27]:
```
testing_padded[0]
```

Out[27]:
```
array([2496,   30,    4,   12,   27,    1,    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])
```

In [28]:
```
# Reverse the word_index
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
reverse_word_index[1084]
```

Out[28]:
```
'announce'
```

In [29]:
```
# Decode the sentence
def decode_sentence(text):
    word=list()
    for i in text:
        if i != 0:
            word.append(reverse_word_index[i])
    sentence = ' '.join(word)
    return 'sentence :',sentence
```

In [30]:

```
decode_sentence(testing_padded[0])
```

Out[30]:

```
('sentence :', 'glasses are the new it <OOV>')
```

In [31]:

```
y_test.head(1)
```

Out[31]:

```
10465    0
Name: is_sarcastic, dtype: int64
```

In [32]:

```
e = model.layers[0]
e
```

Out[32]:

```
<keras.layers.embeddings.Embedding at 0x2b5be807820>
```

In [33]:

```
weights = e.get_weights()[0]
weights[0]
```

Out[33]:

```
array([ 0.1600546 ,  0.01401678,  0.00055776,  0.04858763,  0.0108338 ,
       -0.12296436, -0.0004041 ,  0.3463165 ,  0.03926046, -0.01363823,
       -0.04651956, -0.03632296, -0.00501014, -0.05600563,  0.07940689,
       -0.0232128 ], dtype=float32)
```

In [34]:

```
vocab_size, embedding_dim = weights.shape
vocab_size
```

Out[34]:

```
10000
```

Save tokenize set & Predicted value

In [35]:

```
import io

out_v = io.open('vecs.tsv', 'w', encoding='utf-8')
out_m = io.open('meta.tsv', 'w', encoding='utf-8')
for word_num in range(1, vocab_size):
    word = reverse_word_index[word_num]
    embeddings = weights[word_num]
    out_m.write(word + "\n")
    out_v.write('\t'.join([str(x) for x in embeddings]) + "\n")
out_v.close()
out_m.close()
```

In [36]:

```
sentence = ["No matter who wins, This election will seriosly boost alcohal sales."]
sequences = tokenizer.texts_to_sequences(sentence)
padded = pad_sequences(sequences, maxlen=100, padding='post', truncating='post')
np.argmax(model.predict(padded))
```

Out[36]:

```
0
```

In [37]:

```
sentence = ["I work 40 hours a week for me to be this poor!"]
sequences = tokenizer.texts_to_sequences(sentence)
padded = pad_sequences(sequences, maxlen=100, padding='post', truncating='post')
np.argmax(model.predict(padded))
```

Out[37]:

```
0
```

In [38]:

```
sentence = ["Well, what a surprise."]
sequences = tokenizer.texts_to_sequences(sentence)
padded = pad_sequences(sequences, maxlen=100, padding='post', truncating='post')
np.argmax(model.predict(padded))
```

Out[38]:

```
0
```

In [39]:

```
sentence = ['I will consider myself lucky to die alone']
sequences = tokenizer.texts_to_sequences(sentence)
padded = pad_sequences(sequences, maxlen=100, padding='post', truncating='post')
np.argmax(model.predict(padded))
```

Out[39]:

```
0
```

In [40]:

```
sentence = ["Friday is my second favorite f-word"]
sequences = tokenizer.texts_to_sequences(sentence)
padded = pad_sequences(sequences, maxlen=100, padding='post', truncating='post')
np.argmax(model.predict(padded))
```

Out[40]:

```
0
```

In [41]:

```
sentence = ['Hope followers have increased!']
sequences = tokenizer.texts_to_sequences(sentence)
```

```
padded = pad_sequences(sequences, maxlen=100, padding='post', truncating='post')
np.argmax(model.predict(padded))
```

Out[41]:

```
0
```

In [42]:

```
sentence = ['''So let it be with Caesar.
The noble Brutus / Hath told you Caesar was ambitious.
If it were so, it was a grievous fault,
And grievously hath Caesar answered it.''']
sequences = tokenizer.texts_to_sequences(sentence)
padded = pad_sequences(sequences, maxlen=100, padding='post', truncating='post')
np.argmax(model.predict(padded))
```

Out[42]:

```
0
```