Configuration Manual

MSc Research Project
Data Analytics

Shubham Gosavi
Student ID: x20190824

School of Computing
National College of Ireland

Supervisor: Amandeep Singh

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	SHUBHAM RAM GOSAVI
Student ID:	X20190824
Programme:	MSc. Data Analytics — **Year:** 2021 - 2022
Module:	Research Project
Lecturer:	Amandeep Singh
Submission Due Date:	19th September 2022
Project Title:	Transformer based Detection of Sarcasm and it's Sentiment in Textual Data
Word Count:	1136

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Shubham Ram Gosavi

Date: 12/09/2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Shubham Gosavi
Student ID: x20198024

1 Software Description

Programming Tools	Jupyter Notebook, Google Collaboratory, Python, Python Libraries, Overleaf (for report).
Browser	Opera, Google Crome
Mailing address	Gmail, google drive (to retrieve data)

Table 1. Software Description

2 Environment Setup

"Jupyter Notebook" and "Google Colab" computing platforms was chosen for the implementation of machine learning and deep learning models. Jupyter notebook being an OS based software runs on the ram and GPU present in the system hence is very reliable to access the program anytime and start from wherever saved. Whereas Google Colab is a browser-based software which is owned by google and is best when modelling high configuration models as it provides free access to cloud GPU which makes the implementation faster. Hence, both the platforms were selected.
Making a Jupyter Notebook sheet:

1. Download and open Jupyter notebook and select the latest python environment which in this case is python 3.

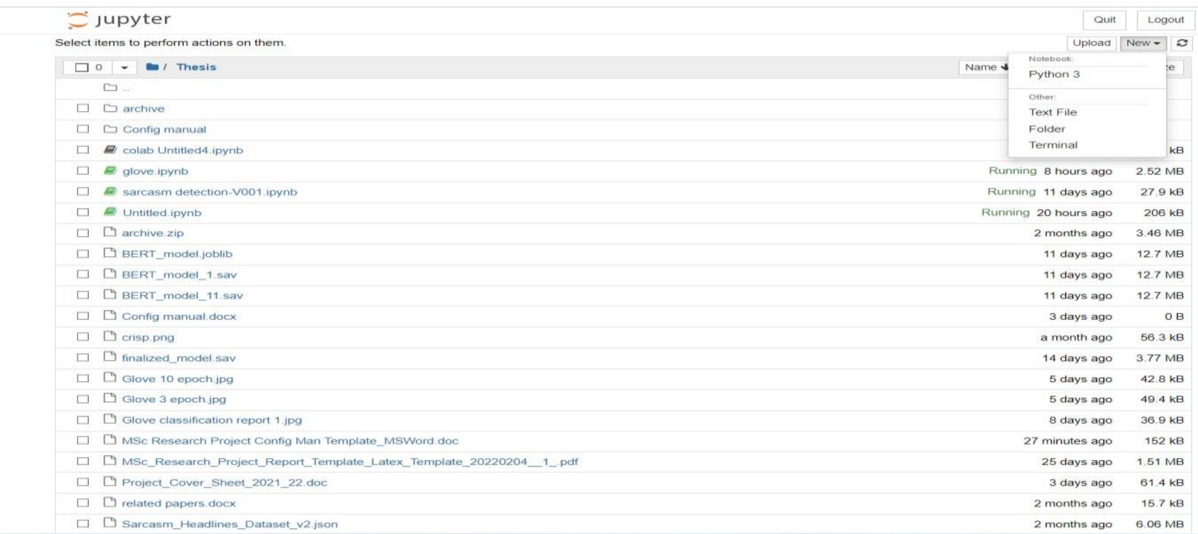

Figure 1. Jupyter Notebook Environment

2. Choose a folder and save the dataset which is going to be imported in the python environment.
3. Click on new on the right top corner and open python 3 notebook.

3 News Headline dataset.

3.1 Data Preparation.

1. In order to import the dataset in the python environment first the dataset "*News Headline dataset for Sarcasm detection*" (Mishra, 2019) is supposed to be downloaded from website "kaggle.com" (Mishra, News Headlines Dataset For Sarcasm Detection, 2019)

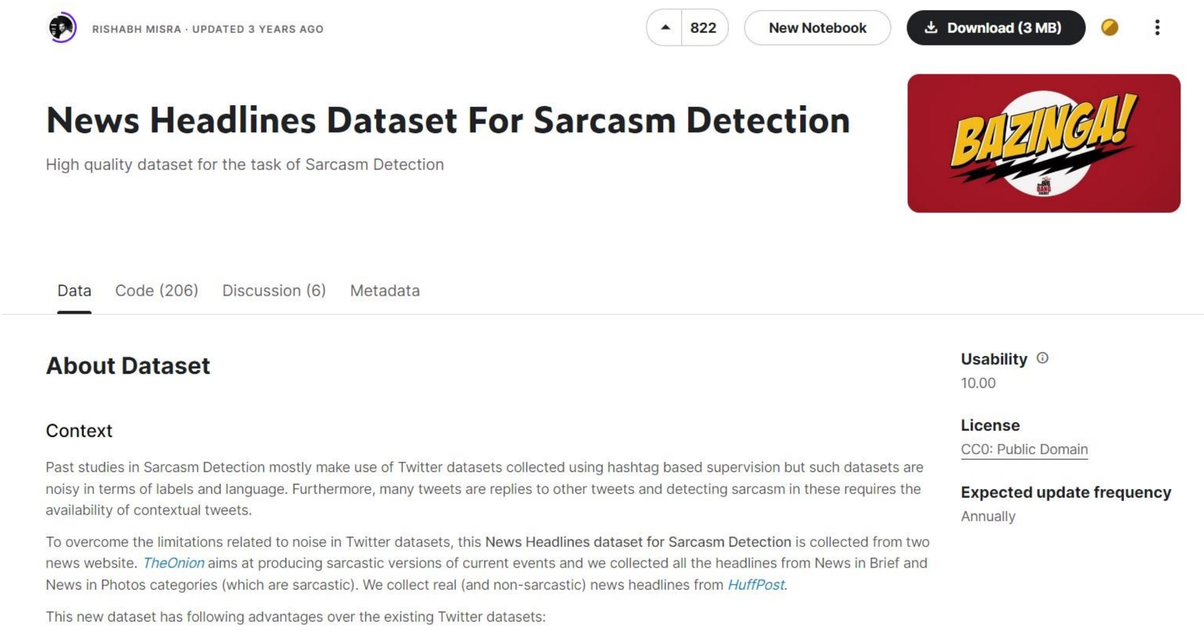

Figure 2 News Headline Dataset

2. Once downloaded upload the dataset into your google drive and will be used later when working with google colab.
3. Import the libraries that are mentioned in the figure below.

In [2]:
```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
from sklearn.preprocessing import LabelBinarizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from wordcloud import WordCloud,STOPWORDS
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize,sent_tokenize
from bs4 import BeautifulSoup
import re,string,unicodedata
from keras.preprocessing import text, sequence
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
from sklearn.model_selection import train_test_split
from string import punctuation
import keras
from keras.models import Sequential
from keras.layers import Dense,Embedding,LSTM,Dropout,Bidirectional,GRU
import tensorflow as tf
import pickle
```

Figure 3. Python Libraries

4. Import the dataset in the python 3 environment and view first few rows using the below command.

In [3]:
```python
df = pd.read_json("Sarcasm_Headlines_Dataset_v2.json", lines=True)
df.head()
```

Out[3]:

	is_sarcastic	headline	article_link
0	1	thirtysomething scientists unveil doomsday clo...	https://www.theonion.com/thirtysomething-scien...
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...

3.2 Data Pre-processing and cleaning.

1. A new function named "*Clean_text*" is created to pass the text from the data as its parameter and clean the entire text.

In [7]:
```python
def clean_text(text):

    ### Text to lower case
    text = text.lower()

    ### Step 1 remove Hashtags
    pattern_one = "#[A-z0-9_\-\.#\$%]{1,}"
    text = re.sub(pattern_one,"",text)

    ### Step 2 Remove Mentions
    pattern_two = "@[A-z0-9_\-\.#\$%]{1,}"
    text = re.sub(pattern_two,"",text)

    ### Step 3 Remove Urls
    url = "https?://[A-z0-9_\-\.#\$%\?=&]+(/[A-z0-9_\-\.#\$%\?=&]+)*"
    text = re.sub(url,"",text)
    ### Step 4 remove numbers
    numbers = "\d+"
    text = re.sub(numbers,"",text)
    return text

df['headline'] = df["headline"].apply(clean_text)
```

2. Now that the data is cleaned there are more than 10,000 words in the data. Therefore, we create a word cloud to view the most frequently used sarcastic words and non-sarcastic words.

```
In [9]:
#word cloud for non sarcastic non label


plt.figure(figsize = (20,20)) # Text that is Sarcastic
wc = WordCloud(max_words = 2000 , width = 1600 , height = 800).generate(" ".join(df[df.is_sarcastic == 1].headline))
plt.imshow(wc , interpolation = 'bilinear')


Out[9]: <matplotlib.image.AxesImage at 0x1c197ca4888>
```

```
In [8]:
#word cloud for non sarcastic label

plt.figure(figsize = (20,20)) # Text that is Not Sarcastic
wc = WordCloud(max_words = 2000 , width = 1600 , height = 800).generate(" ".join(df[df.is_sarcastic == 0].headline))
plt.imshow(wc , interpolation = 'bilinear')


Out[8]: <matplotlib.image.AxesImage at 0x1c197b10188>
```

Figure 4. Word Cloud

3. Next, for the word to vec model using a for loop all the words from the sentences are split into unique words and stored as a list named "*word*" and using genism library a word to vec model is created passing the "*word*" list are its parameter. Words tokenization is also done to the same list which will be used for modelling.

```
In [10]:
words = []
for i in df.headline.values:
    words.append(i.split())
words[:5]
```

```
In [11]:
import gensim
#Dimension of vectors we are generating
EMBEDDING_DIM = 200

#Creating Word Vectors by Word2Vec Method
w2v_model = gensim.models.Word2Vec(sentences = words , size=EMBEDDING_DIM , window = 5 , min_count = 1)
```

```
In [12]:


#vocab size
len(w2v_model.wv.vocab)
#We have now represented each of 37149 words by a 100dim vector.
```

```
In [13]:
tokenizer = text.Tokenizer(num_words=37149)
tokenizer.fit_on_texts(words)
tokenized_train = tokenizer.texts_to_sequences(words)
x = tf.keras.utils.pad_sequences(tokenized_train, maxlen = 20)
```

Figure 5. Pre-processing for word2vec model.

4. A function named "*get_weight_matrix*" is created which creates a weighted matrix from the word2vec genism model and using it as weights of non-trainable keras embedding layer.

```
In [15]:
# Function to create weight matrix from word2vec gensim model
def get_weight_matrix(model, vocab):
    # total vocabulary size plus 0 for unknown words
    vocab_size = len(vocab) + 1
    # define weight matrix dimensions with all 0
    weight_matrix = np.zeros((vocab_size, EMBEDDING_DIM))
    # step vocab, store vectors using the Tokenizer's integer mapping
    for word, i in vocab.items():
        weight_matrix[i] = model[word]
    return weight_matrix
```

```
In [16]:
#Getting embedding vectors from word2vec and usings it as weights of non-trainable keras embedding layer
embedding_vectors = get_weight_matrix(w2v_model, tokenizer.word_index)
```

Figure 6. Weight Matrix function.

3.3 Modelling

1. Import the train test split library from Sklearn and split the data into train and test with split ratio being 70:30 between training and validation respectively.

In [19]:
```
x_train, x_test, y_train, y_test = train_test_split(x, df.is_sarcastic , test_size = 0.3 , random_state = 0)
```

3.3.1 For GloVe model

1. Initialize the neural network model store in variable "*model*" and add the layers as shown in the figure below and view how the model will look.

In [17]:
```
#Defining Neural Network
model = Sequential()
#Non-trainable embeddidng layer
model.add(Embedding(vocab_size, output_dim=EMBEDDING_DIM, weights=[embedding_vectors], input_length=20, trainable=True))
#LSTM
model.add(Bidirectional(LSTM(units=128 , recurrent_dropout = 0.3 , dropout = 0.3,return_sequences = True)))
model.add(Bidirectional(GRU(units=32 , recurrent_dropout = 0.1 , dropout = 0.1)))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer=keras.optimizers.Adam(lr = 0.01), loss='binary_crossentropy', metrics=['acc'])

del embedding_vectors
```

In [18]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 20, 200)           7430000

 bidirectional (Bidirectiona  (None, 20, 256)          336896
 l)

 bidirectional_1 (Bidirectio  (None, 64)               55680
 nal)

 dense (Dense)               (None, 1)                 65

=================================================================
Total params: 7,822,641
Trainable params: 7,822,641
Non-trainable params: 0
_________________________________________________________________
```

Figure 7. Neural Network for word2vec model

2. Train the model as shown.

In [23]:
```
history = model.fit(x_train, y_train, batch_size = 128 , validation_data = (x_test,y_test) , epochs = 10)
```

```
Epoch 1/10
157/157 [==============================] - 54s 342ms/step - loss: 0.0063 - acc: 0.9984 - val_loss: 0.8013 - val_acc: 0.8424
Epoch 2/10
157/157 [==============================] - 48s 303ms/step - loss: 0.0023 - acc: 0.9993 - val_loss: 0.8262 - val_acc: 0.8385
Epoch 3/10
157/157 [==============================] - 47s 301ms/step - loss: 0.0012 - acc: 0.9996 - val_loss: 1.0142 - val_acc: 0.8414
Epoch 4/10
157/157 [==============================] - 48s 304ms/step - loss: 0.0021 - acc: 0.9994 - val_loss: 1.1385 - val_acc: 0.8338
Epoch 5/10
157/157 [==============================] - 47s 300ms/step - loss: 0.0035 - acc: 0.9990 - val_loss: 1.0105 - val_acc: 0.8298
Epoch 6/10
157/157 [==============================] - 45s 284ms/step - loss: 0.0014 - acc: 0.9995 - val_loss: 1.3567 - val_acc: 0.8038
Epoch 7/10
157/157 [==============================] - 47s 302ms/step - loss: 0.0018 - acc: 0.9995 - val_loss: 1.1499 - val_acc: 0.8368
Epoch 8/10
157/157 [==============================] - 48s 303ms/step - loss: 5.9538e-04 - acc: 0.9999 - val_loss: 1.2429 - val_acc: 0.8334
Epoch 9/10
157/157 [==============================] - 50s 319ms/step - loss: 7.5379e-04 - acc: 0.9998 - val_loss: 1.3153 - val_acc: 0.8298
Epoch 10/10
157/157 [==============================] - 47s 301ms/step - loss: 0.0027 - acc: 0.9992 - val_loss: 1.2227 - val_acc: 0.8181
```

3. Plot the results as shown below.

In [29]:
```
epochs = [i for i in range(10)]
fig , ax = plt.subplots(1,2)
train_acc = history.history['acc']
train_loss = history.history['loss']
val_acc = history.history['val_acc']
val_loss = history.history['val_loss']
fig.set_size_inches(20,10)

ax[0].plot(epochs , train_acc , 'go-' , label = 'Training Accuracy')
ax[0].plot(epochs , val_acc , 'ro-' , label = 'Testing Accuracy')
ax[0].set_title('Training & Testing Accuracy')
ax[0].legend()
ax[0].set_xlabel("Epochs")
ax[0].set_ylabel("Accuracy")

ax[1].plot(epochs , train_loss , 'go-' , label = 'Training Loss')
ax[1].plot(epochs , val_loss , 'ro-' , label = 'Testing Loss')
ax[1].set_title('Training & Testing Loss')
ax[1].legend()
ax[1].set_xlabel("Epochs")
ax[1].set_ylabel("Loss")
plt.show()
```

Figure 8. w2v plot code

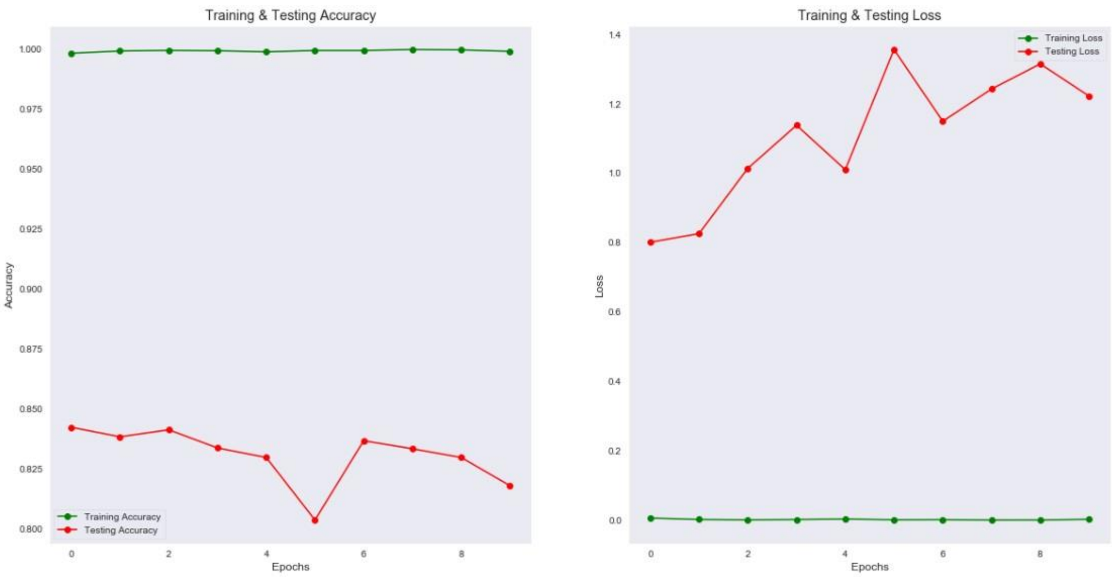

Figure 9. Word2Vec results plot

4. Run the code below to predict on test data and create a confusion matrix and classification report.

In [ ]:
```
predict_x=model.predict(x_test)
```

In [ ]:
```
predictions = (model.predict(x_test) > 0.5).astype("int32")
```

In [ ]:
```
cm = confusion_matrix(y_test,predictions)
cm
```

In [ ]:
```
print(classification_report(y_test, predictions, target_names = ['Not Sarcastic','Sarcastic']))
```

3.3.2 For BERT (10 Epoch) Model.

1. Import all the required libraries and the pre trained model from the keras library and run as shown in figure below (Martín Abadi, 2015).

BERT ¶

```
In [30]:
import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_text as text
```

```
In [31]:
bert_preprocess = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3")
bert_encoder = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/4")
```

Figure 10. Loading BERT libraries.

2. Create a function named "*get_sentence_embedding*" with sentence as its parameter.

```
In [ ]:
def get_sentence_embeding(sentences):
    preprocessed_text = bert_preprocess(sentences)
    return bert_encoder(preprocessed_text)['pooled_output']

```

Figure 11. Function to get sentence embedding

3. Initialize a neural network, include layers as shown below and compile.

```
In [33]:
# Bert layers
text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
preprocessed_text = bert_preprocess(text_input)
outputs = bert_encoder(preprocessed_text)

# Neural network layers
l = tf.keras.layers.Dropout(0.1, name="dropout")(outputs['pooled_output'])
l = tf.keras.layers.Dense(1, activation='sigmoid', name="output")(l)

# Use inputs and outputs to construct a final model
model = tf.keras.Model(inputs=[text_input], outputs = [l])

```

Figure 12. BERT Model

4. Include metrics to print in the output.

```
In [35]:
METRICS = [
      tf.keras.metrics.BinaryAccuracy(name='accuracy'),
      tf.keras.metrics.Precision(name='precision'),
      tf.keras.metrics.Recall(name='recall')
]

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=METRICS)


```

Figure 13. Metrics

5. Model Summary.

```
In [34]:  1  model.summary()

Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 text (InputLayer)              [(None,)]            0           []

 keras_layer (KerasLayer)       {'input_mask': (Non  0           ['text[0][0]']
                                e, 128),
                                 'input_type_ids':
                                (None, 128),
                                 'input_word_ids':
                                (None, 128)}

 keras_layer_1 (KerasLayer)     {'default': (None,   109482241   ['keras_layer[0][0]',
                                768),                             'keras_layer[0][1]',
                                 'encoder_outputs':               'keras_layer[0][2]']
                                 [(None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768)],
                                 'pooled_output': (
                                None, 768),
                                 'sequence_output':
                                 (None, 128, 768)}

 dropout (Dropout)              (None, 768)          0           ['keras_layer_1[0][13]']

 output (Dense)                 (None, 1)            769         ['dropout[0][0]']

==================================================================================================
Total params: 109,483,010
Trainable params: 769
Non-trainable params: 109,482,241
__________________________________________________________________________________________________
```

Figure 14. BERT Model Summary

6. Create Early Stop variable to pass early stopping function.

```
In [38]:
earlystop = tf.keras.callbacks.EarlyStopping(
    monitor='accuracy', patience=3,
    mode='max')
```

7. Fit the BERT model and store in variable “*h1*”.

```
[23] h1 = model.fit(x_train, y_train, epochs=10, validation_data = (x_test,y_test), callbacks=[earlystop])

Epoch 1/10
627/627 [==============================] - 300s 454ms/step - loss: 0.6085 - accuracy: 0.6771 - precision: 0.6836 - recall: 0.6082 - val_loss: 0.5332 - val_accuracy: 0.7660 - val_precision: 0.7706 - val_recall: 0.7134
Epoch 2/10
627/627 [==============================] - 285s 454ms/step - loss: 0.5409 - accuracy: 0.7388 - precision: 0.7452 - recall: 0.6920 - val_loss: 0.5016 - val_accuracy: 0.7767 - val_precision: 0.7645 - val_recall: 0.7571
Epoch 3/10
627/627 [==============================] - 285s 454ms/step - loss: 0.5182 - accuracy: 0.7524 - precision: 0.7547 - recall: 0.7167 - val_loss: 0.5164 - val_accuracy: 0.7283 - val_precision: 0.8573 - val_recall: 0.5045
Epoch 4/10
627/627 [==============================] - 285s 455ms/step - loss: 0.5089 - accuracy: 0.7562 - precision: 0.7572 - recall: 0.7238 - val_loss: 0.4786 - val_accuracy: 0.7749 - val_precision: 0.8025 - val_recall: 0.6895
Epoch 5/10
627/627 [==============================] - 285s 454ms/step - loss: 0.4986 - accuracy: 0.7637 - precision: 0.7645 - recall: 0.7331 - val_loss: 0.4658 - val_accuracy: 0.7955 - val_precision: 0.7783 - val_recall: 0.7884
Epoch 6/10
627/627 [==============================] - 285s 454ms/step - loss: 0.4956 - accuracy: 0.7686 - precision: 0.7667 - recall: 0.7438 - val_loss: 0.4598 - val_accuracy: 0.7932 - val_precision: 0.8148 - val_recall: 0.7233
Epoch 7/10
627/627 [==============================] - 284s 453ms/step - loss: 0.4929 - accuracy: 0.7668 - precision: 0.7670 - recall: 0.7381 - val_loss: 0.5003 - val_accuracy: 0.7404 - val_precision: 0.8752 - val_recall: 0.5206
Epoch 8/10
627/627 [==============================] - 284s 454ms/step - loss: 0.4909 - accuracy: 0.7666 - precision: 0.7666 - recall: 0.7380 - val_loss: 0.4575 - val_accuracy: 0.7877 - val_precision: 0.8384 - val_recall: 0.6778
Epoch 9/10
627/627 [==============================] - 285s 454ms/step - loss: 0.4848 - accuracy: 0.7707 - precision: 0.7695 - recall: 0.7451 - val_loss: 0.4698 - val_accuracy: 0.7922 - val_precision: 0.7303 - val_recall: 0.8830
Epoch 10/10
627/627 [==============================] - 284s 454ms/step - loss: 0.4844 - accuracy: 0.7703 - precision: 0.7679 - recall: 0.7467 - val_loss: 0.4705 - val_accuracy: 0.7888 - val_precision: 0.7221 - val_recall: 0.8937
```

Figure 15. BERT with 10 epochs

8. Plot the results of above created model.

```python
epochs = [i for i in range(10)]
fig , ax = plt.subplots(1,2)
train_acc = h1.history['accuracy']
train_loss = h1.history['loss']
val_acc = h1.history['val_accuracy']
val_loss = h1.history['val_loss']
fig.set_size_inches(20,10)

ax[0].plot(epochs , train_acc , 'go-' , label = 'Training Accuracy')
ax[0].plot(epochs , val_acc , 'ro-' , label = 'Testing Accuracy')
ax[0].set_title('Training & Testing Accuracy')
ax[0].legend()
ax[0].set_xlabel("Epochs")
ax[0].set_ylabel("Accuracy")

ax[1].plot(epochs , train_loss , 'go-' , label = 'Training Loss')
ax[1].plot(epochs , val_loss , 'ro-' , label = 'Testing Loss')
ax[1].set_title('Training & Testing Loss')
ax[1].legend()
ax[1].set_xlabel("Epochs")
ax[1].set_ylabel("Loss")
plt.show()
```

Figure 16. BERT results plot code.

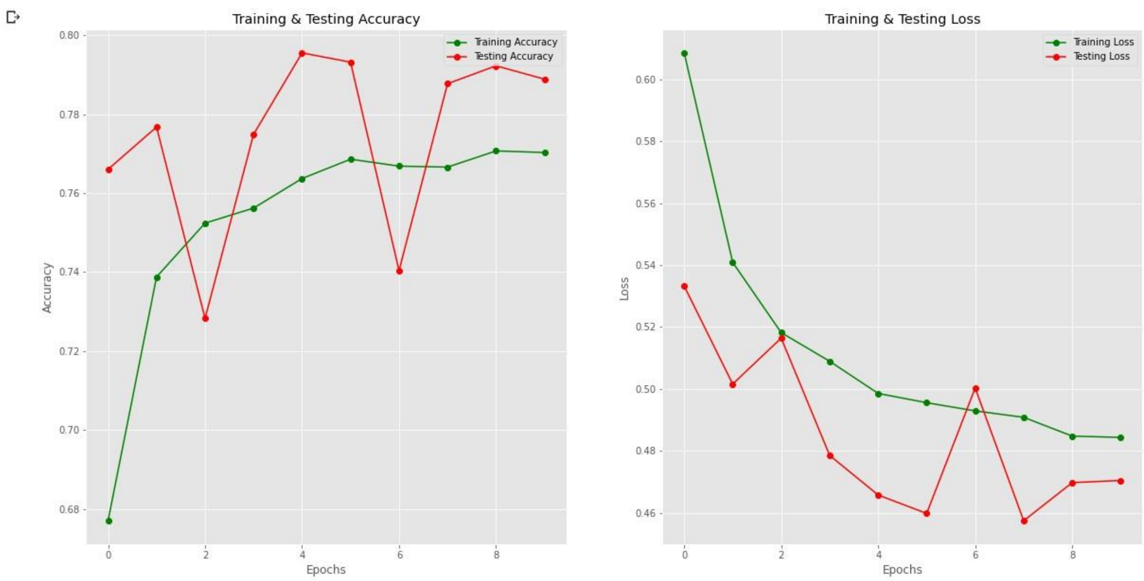

Figure 17. BERT Result Plot (10 Epoch).

3.3.3 For BERT (20 Epoch) Model.

1. Following the same implementation as above run the BERT model for 20 epochs (Jacob Devlin, 2018).

```
[24] h1 = model.fit(x_train, y_train, epochs=20, validation_data = (x_test,y_test), callbacks=[earlystop])

Epoch 1/20
627/627 [==============================] - 296s 453ms/step - loss: 0.6043 - accuracy: 0.6784 - precision: 0.6840 - recall: 0.6125 - val_loss: 0.5307 - val_accuracy: 0.7681 - val_precision: 0.7723 - val_recall: 0.7168
Epoch 2/20
627/627 [==============================] - 287s 458ms/step - loss: 0.5366 - accuracy: 0.7438 - precision: 0.7484 - recall: 0.7017 - val_loss: 0.5010 - val_accuracy: 0.7795 - val_precision: 0.7485 - val_recall: 0.7978
Epoch 3/20
627/627 [==============================] - 283s 452ms/step - loss: 0.5171 - accuracy: 0.7550 - precision: 0.7565 - recall: 0.7213 - val_loss: 0.4914 - val_accuracy: 0.7798 - val_precision: 0.7298 - val_recall: 0.8420
Epoch 4/20
627/627 [==============================] - 283s 452ms/step - loss: 0.5051 - accuracy: 0.7605 - precision: 0.7590 - recall: 0.7338 - val_loss: 0.4701 - val_accuracy: 0.7892 - val_precision: 0.7933 - val_recall: 0.7444
Epoch 5/20
627/627 [==============================] - 283s 452ms/step - loss: 0.5001 - accuracy: 0.7623 - precision: 0.7640 - recall: 0.7300 - val_loss: 0.4637 - val_accuracy: 0.7907 - val_precision: 0.8133 - val_recall: 0.7186
Epoch 6/20
627/627 [==============================] - 283s 452ms/step - loss: 0.4949 - accuracy: 0.7658 - precision: 0.7655 - recall: 0.7377 - val_loss: 0.4583 - val_accuracy: 0.7947 - val_precision: 0.7908 - val_recall: 0.7643
Epoch 7/20
627/627 [==============================] - 283s 451ms/step - loss: 0.4910 - accuracy: 0.7719 - precision: 0.7724 - recall: 0.7435 - val_loss: 0.4535 - val_accuracy: 0.7986 - val_precision: 0.7932 - val_recall: 0.7717
Epoch 8/20
627/627 [==============================] - 283s 451ms/step - loss: 0.4891 - accuracy: 0.7694 - precision: 0.7703 - recall: 0.7398 - val_loss: 0.4660 - val_accuracy: 0.7913 - val_precision: 0.7331 - val_recall: 0.8726
Epoch 9/20
627/627 [==============================] - 283s 452ms/step - loss: 0.4842 - accuracy: 0.7729 - precision: 0.7697 - recall: 0.7514 - val_loss: 0.4747 - val_accuracy: 0.7824 - val_precision: 0.7131 - val_recall: 0.8969
Epoch 10/20
627/627 [==============================] - 283s 451ms/step - loss: 0.4826 - accuracy: 0.7753 - precision: 0.7738 - recall: 0.7510 - val_loss: 0.4429 - val_accuracy: 0.8063 - val_precision: 0.8070 - val_recall: 0.7715
Epoch 11/20
627/627 [==============================] - 282s 450ms/step - loss: 0.4808 - accuracy: 0.7757 - precision: 0.7756 - recall: 0.7492 - val_loss: 0.4476 - val_accuracy: 0.8027 - val_precision: 0.7636 - val_recall: 0.8390
Epoch 12/20
627/627 [==============================] - 282s 451ms/step - loss: 0.4789 - accuracy: 0.7774 - precision: 0.7754 - recall: 0.7543 - val_loss: 0.4437 - val_accuracy: 0.8064 - val_precision: 0.7741 - val_recall: 0.8291
Epoch 13/20
627/627 [==============================] - 282s 450ms/step - loss: 0.4786 - accuracy: 0.7779 - precision: 0.7747 - recall: 0.7571 - val_loss: 0.4431 - val_accuracy: 0.7980 - val_precision: 0.8339 - val_recall: 0.7109
Epoch 14/20
627/627 [==============================] - 287s 459ms/step - loss: 0.4795 - accuracy: 0.7718 - precision: 0.7705 - recall: 0.7465 - val_loss: 0.4427 - val_accuracy: 0.8029 - val_precision: 0.7686 - val_recall: 0.8294
Epoch 15/20
627/627 [==============================] - 289s 460ms/step - loss: 0.4784 - accuracy: 0.7749 - precision: 0.7725 - recall: 0.7521 - val_loss: 0.4412 - val_accuracy: 0.8044 - val_precision: 0.7648 - val_recall: 0.8418
Epoch 16/20
627/627 [==============================] - 282s 450ms/step - loss: 0.4752 - accuracy: 0.7774 - precision: 0.7756 - recall: 0.7542 - val_loss: 0.4341 - val_accuracy: 0.8114 - val_precision: 0.7962 - val_recall: 0.8035
```

Figure 18. BERT Model Implementation

2. Plot the results of above created model.

```
[26] epochs = [i for i in range(16)]
     fig , ax = plt.subplots(1,2)
     train_acc = h1.history['accuracy']
     train_loss = h1.history['loss']
     val_acc = h1.history['val_accuracy']
     val_loss = h1.history['val_loss']
     fig.set_size_inches(20,10)

     ax[0].plot(epochs , train_acc , 'go-' , label = 'Training Accuracy')
     ax[0].plot(epochs , val_acc , 'ro-' , label = 'Validation Accuracy')
     ax[0].set_title('Training & Validation Accuracy')
     ax[0].legend()
     ax[0].set_xlabel("Epochs")
     ax[0].set_ylabel("Accuracy")

     ax[1].plot(epochs , train_loss , 'go-' , label = 'Training Loss')
     ax[1].plot(epochs , val_loss , 'ro-' , label = 'Validation Loss')
     ax[1].set_title('Training & Validation Loss')
     ax[1].legend()
     ax[1].set_xlabel("Epochs")
     ax[1].set_ylabel("Loss")
     plt.show()
```

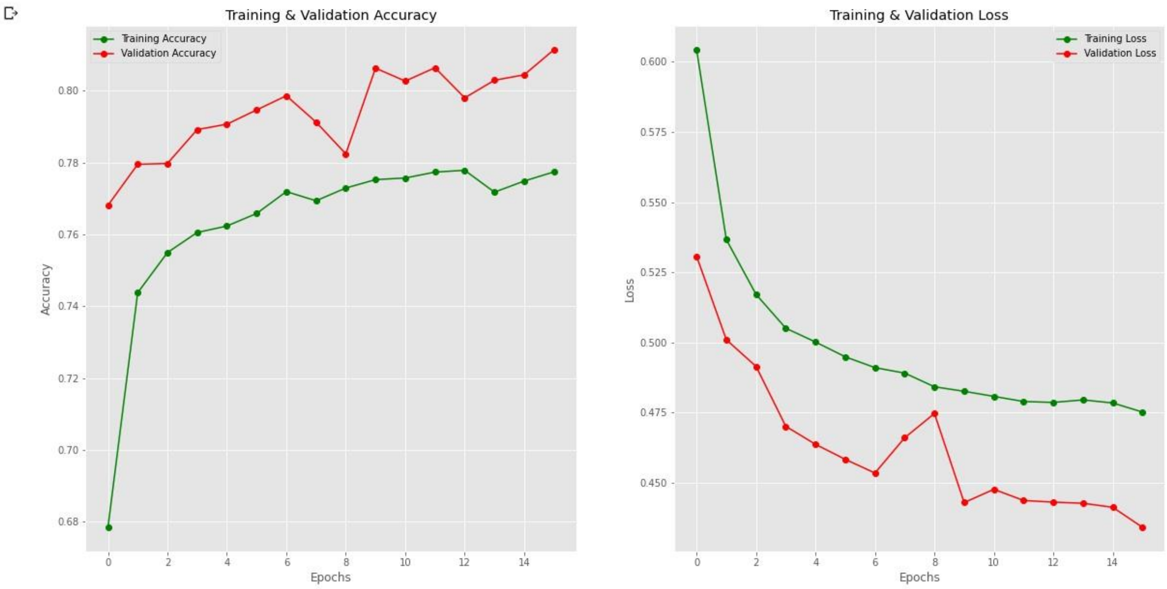

Figure 19.BERT Result Plot (20 Epoch).

3. As the BERT model with 20 epoch gives more accuracy this model will be used to predict on the test data and generate classification report. Run the code below to test the model on the test data.

```
In [ ]: test_df=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Test_datat - Copy.csv")
```

```
In [ ]: y_predicted = model.predict(test_df) > 0.5
```

```
In [ ]: Y_test_datat=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Y_test_datat.csv")
```

```
In [ ]: mat = confusion_matrix(Y_test_datat, y_predicted)
        labels = ['Sarcastic ', 'non-sarcastic']
```

```
In [ ]: sns.heatmap(mat, square=True, annot=True, fmt='d', cbar=False, cmap='Blues',
                    xticklabels=labels, yticklabels=labels)

        plt.xlabel('Predicted label')
        plt.ylabel('Actual label')
```

```
In [ ]: print(classification_report(Y_test_datat, y_predicted))
```

Figure 20.Model Testing and Classification Report

3.3.4 VADER

```
In [17]: import re
         import string
         import nltk
         import matplotlib.pyplot as plt
         import seaborn as sns
         sns.set_style('darkgrid')

         nltk.download('vader_lexicon')
         from nltk.sentiment.vader import SentimentIntensityAnalyzer as SIA
         from wordcloud import WordCloud,STOPWORDS

         plt.rc('figure',figsize=(17,13))
         import plotly.express as px
         import plotly.graph_objs as go
         import plotly.offline as pyo
         from plotly.subplots import make_subplots

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

Figure 21. Libraries to Implement VADER

In [18]:

```
data['headline'] = data['headline']
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
analyser = SentimentIntensityAnalyzer()
scores=[]
for i in range(len(data['headline'])):

    score = analyser.polarity_scores(data['headline'][i])
    score=score['compound']
    scores.append(score)
sentiment=[]
for i in scores:
    if i>=0.05:
        sentiment.append('Positive')
    elif i<=(-0.05):
        sentiment.append('Negative')
    else:
        sentiment.append('Neutral')
data['sentiment']=pd.Series(np.array(sentiment))
```

Figure 22. VADER Implementation.

In [56]:

```
data.head(10)
```

Out[56]:

	is_sarcastic	headline	sentiment
0	1	thirtysomething scientists unveil doomsday clo...	Negative
1	0	dem rep. totally nails why congress is falling...	Negative
2	0	eat your veggies: 9 deliciously different recipes	Positive
3	1	inclement weather prevents liar from getting t...	Negative
4	1	mother comes pretty close to using word 'strea...	Positive
5	0	my white inheritance	Neutral
6	0	5 ways to file your taxes with less stress	Negative
7	1	richard branson's global-warming donation near...	Negative
8	1	shadow government getting too large to meet in...	Neutral
9	0	lots of parents know this scenario	Neutral

Figure 23. VADER Result.

```
In [58]:
#sarcasm wise negative sentiment count
negative_count = data.groupby('is_sarcastic')['sentiment'].apply(lambda x: (x=='Negative').sum()).reset_index(name='Negative_count')

#view results
print(negative_count)
```

```
   is_sarcastic  Negative_count
0             0            4781
1             1            4653
```

```
In [43]:
#sarcasm wise positive sentiment count
positive_count = data.groupby('is_sarcastic')['sentiment'].apply(lambda x: (x=='Positive').sum()).reset_index(name='Positive_count')

#view results
print(positive_count)
```

```
   is_sarcastic  Positive_count
0             0            4668
1             1            4271
```

```
In [44]:
#sarcasm wise neutral sentiment count
neutral_count = data.groupby('is_sarcastic')['sentiment'].apply(lambda x: (x=='Neutral').sum()).reset_index(name='neutral_count')

#view results
print(neutral_count)
```

```
   is_sarcastic  neutral_count
0             0           5536
1             1           4710
```

Figure 24. VADER group by Result

References

Jacob Devlin, M.-}. C. (2018). Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*.

Martín Abadi, A. A. (2015). {TensorFlow}: Large-Scale Machine Learning on Heterogeneous Systems. *Software available from tensorflow.org*.

Mishra, R. (2019). *News Headlines Dataset For Sarcasm Detection*. Retrieved from Kaggle: https://www.kaggle.com/datasets/rmisra/news-headlines-dataset-for-sarcasm-detection

Mishra, R. (2019). Sarcasm Detection using Hybrid Neural Network. *CoRR*.