

Jason  
2401960183  
Deep Learning Final Exam No 2

## Importing Libraries

```
import numpy as np
import pandas as pd
import nltk
import string as s
import re
import matplotlib.pyplot as plt
import os
import string
import tensorflow
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Embedding, LSTM, Dense, Input, Dropout
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.metrics import classification_report, f1_score, precision_score, recall_score, accuracy_score
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
True
```

## 1. Loading and Preprocessing Data using NLP Methods

### Loading the clickbait.csv dataset

```
data = pd.read_csv("./clickbait.csv")
data.head()
```

	headline	clickbait
0	Should I Get Bings	1
1	Which TV Female Friend Group Do You Belong In	1
2	The New "Star Wars: The Force Awakens" Trailer...	1
3	This Vine Of New York On "Celebrity Big Brothe...	1
4	A Couple Did A Stunning Photo Shoot With Their...	1

```
data.shape
```

```
(32000, 2)
```

### Get the features (X) and the labels (y)

```
X = data['headline']
y = data['clickbait']
```

```
print(X.head())
```

```
0          Should I Get Bings
1  Which TV Female Friend Group Do You Belong In
2  The New "Star Wars: The Force Awakens" Trailer...
3  This Vine Of New York On "Celebrity Big Brothe...
4  A Couple Did A Stunning Photo Shoot With Their...
Name: headline, dtype: object
```

### Function to lowercase the words

```
def lowercase(text):
    return text.lower()
```

```
X = X.apply(lowercase)

print(X.head())

0          should i get bings
1    which tv female friend group do you belong in
2    the new "star wars: the force awakens" trailer...
3    this vine of new york on "celebrity big brothe...
4    a couple did a stunning photo shoot with their...
Name: headline, dtype: object
```

Tokenize or split the main string into an array of strings

```
def tokenization(words):
    list_string = words.split()
    return list_string

X = X.apply(tokenization)

print(X.head())

0          [should, i, get, bings]
1    [which, tv, female, friend, group, do, you, be...
2    [the, new, "star, wars:, the, force, awakens",...
3    [this, vine, of, new, york, on, "celebrity, bi...
4    [a, couple, did, a, stunning, photo, shoot, wi...
Name: headline, dtype: object
```

Removing stopwords

```
stop_words = set(stopwords.words("english"))
def remove_stopwords(words):
    new_words=[]
    for text in words:
        if text not in stop_words:
            new_words.append(text)
    return new_words

X = X.apply(remove_stopwords)

print(X.head())

0          [get, bings]
1    [tv, female, friend, group, belong]
2    [new, "star, wars:, force, awakens", trailer, ...
3    [vine, new, york, "celebrity, big, brother", f...
4    [couple, stunning, photo, shoot, baby, learnin...
Name: headline, dtype: object
```

Remove punctuations on the words

```
def remove_punctuation(words):
    new_words = []
    for text in words:
        for punctuation in string.punctuation:
            text = text.replace(punctuation, '')
        new_words.append(text)
    return new_words

X = X.apply(remove_punctuation)

print(X.head())

0          [get, bings]
1    [tv, female, friend, group, belong]
2    [new, star, wars, force, awakens, trailer, giv...
3    [vine, new, york, celebrity, big, brother, fuc...
4    [couple, stunning, photo, shoot, baby, learnin...
Name: headline, dtype: object
```

Remove numbers and spaces on corresponding words

```
def remove_numbers_and_spaces(words):
    new_words = []
    for text in words:
        result = re.sub(r'\d+', '', text)
        result = re.sub(' +', ' ', result).strip()

        if(result != ""):
            new_words.append(result)
    return new_words

X = X.apply(remove_numbers_and_spaces)

print(X.head())

0          [get, bings]
1    [tv, female, friend, group, belong]
2    [new, star, wars, force, awakens, trailer, giv...
3    [vine, new, york, celebrity, big, brother, fuc...
4    [couple, stunning, photo, shoot, baby, learnin...
Name: headline, dtype: object
```

Split the X and y into train and test

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

Transforms each text in texts to a sequence of integers.

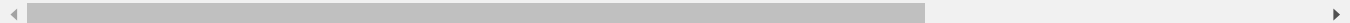
```
vocab_size = 7500
maxlen = 250
embedding_size = 64

tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(X)

X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)

print(X_train[0:5])

[[192, 209, 1216, 3167, 23, 819, 87, 373], [1016, 53, 3122, 6331, 6332, 4835, 710, 695, 1196], [33, 1011, 6979, 788, 243, 499, 99],
```



Pads sequences to the same length so that the model will accept the same dimension of inputs everytime.

```
X_train = pad_sequences(X_train, maxlen=maxlen)
X_test = pad_sequences(X_test, maxlen=maxlen)
```

## 2. Architecture of the NLP Model

For the NLP Model i will be using LSTM, the reason for this is because they can recognize long-term connections in sequential data, which are frequently present in natural language texts, LSTMs are well suited for NLP applications. The meaning of a word frequently depends on the context of the words that come before and after it in this model. These kinds of activities are well suited for LSTMs because of their capacity to store information in memory cells for longer periods of time.

Layers :

1. Embedding layer: This layer takes as input the vocabulary size (vocab\_size), the embedding size (embedding\_size), and the maximum length of the input sequences (maxlen). It transforms the input data into a dense, lower-dimensional representation (embedding) of the input data.
2. LSTM layer: This layer is a Long Short-Term Memory (LSTM) layer with 64 units. It is used to capture the sequential relationships between the words in the input data.
3. Dense layer with ReLU activation: This is a fully connected dense layer with 64 units and a rectified linear unit (ReLU) activation function. It performs non-linear transformations on the output of the LSTM layer.
4. Dense layer with sigmoid activation: This is a fully connected dense layer with 1 unit and a sigmoid activation function. It outputs a probability or a binary classification result.

```
model = Sequential()
model.add(Embedding(vocab_size, embedding_size, input_length=maxlen))
model.add(LSTM(64))
model.add(Dense(64, activation='relu'))
```

```
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 250, 64)	480000
lstm (LSTM)	(None, 64)	33024
dense (Dense)	(None, 64)	4160
dense_1 (Dense)	(None, 1)	65
Total params: 517,249		
Trainable params: 517,249		
Non-trainable params: 0		

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, y_train, batch_size=512, validation_data=(X_test, y_test), epochs=3)
```

```
Epoch 1/3
50/50 [=====] - 10s 83ms/step - loss: 0.5504 - accuracy: 0.8296 - val_loss: 0.2664 - val_accuracy: 0.9495
Epoch 2/3
50/50 [=====] - 2s 40ms/step - loss: 0.1333 - accuracy: 0.9594 - val_loss: 0.1080 - val_accuracy: 0.9597
Epoch 3/3
50/50 [=====] - 2s 36ms/step - loss: 0.0663 - accuracy: 0.9765 - val_loss: 0.1060 - val_accuracy: 0.9583
```

Make predictions using the test set

```
y_predictions = (model.predict(X_test) > 0.5)
```

## 4a. Performance Analysis of the First Model

As we can see from the results of the LSTM NLP Model it has an accuracy of 95% from learning with a batch size of 512 on a 3 epochs run. These results are decent recalling from the metrics below. This architecture is well-suited for problems that involve sequential data and the capture of long-term dependencies. However, it may not be the best choice for problems that do not involve sequential data or where the relationships between elements are not important.

```
print("f1-score: ", f1_score(y_test, y_predictions))
print("precision: ", precision_score(y_test, y_predictions))
print("recall: ", recall_score(y_test, y_predictions))
print("accuracy: ", accuracy_score(y_test, y_predictions))
print("\nClassification Report:\n", classification_report(y_test, y_predictions))
```

```
f1-score: 0.9590553596074222
precision: 0.9519025875190259
recall: 0.9663164400494437
accuracy: 0.95828125
```

```
Classification Report:
              precision    recall  f1-score   support

     0       0.97         0.95         0.96         3164
     1       0.95         0.97         0.96         3236

 accuracy          0.96
 macro avg         0.96
 weighted avg      0.96
```

## 3. Pre Trained NLP Model (Tensorflow BERT)

```
X = data['headline']
y = data['clickbait']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.20)
```

Importing tensorflow hub and text to load the Bert Preprocessor and Bert Encoder before running the model

```
import tensorflow_hub as hub
# !pip3 install --quiet "tensorflow-text==2.8.*"
import tensorflow_text as text
```

```
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")
```

### Making of BERT Pre-Trained Model

```
text_input = Input(shape=(), dtype=tensorflow.string, name='text')
preprocessed_text = bert_preprocess(text_input)
outputs = bert_encoder(preprocessed_text)
```

```
l = Dropout(0.1, name="dropout")(outputs['pooled_output'])
l = Dense(1, activation='sigmoid', name="output")(l)
model = Model(inputs=[text_input], outputs = [l])
```

```
model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
text (InputLayer)	[(None,)]	0	[]
keras_layer (KerasLayer)	{'input_type_ids': (None, 128), 'input_word_ids': (None, 128), 'input_mask': (None, 128)}	0	['text[0][0]']
keras_layer_1 (KerasLayer)	{'encoder_outputs': [(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)], 'default': (None, 768), 'sequence_output': (None, 128, 768), 'pooled_output': (None, 768)}	109482241	['keras_layer[0][0]', 'keras_layer[0][1]', 'keras_layer[0][2]']
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
=====
```

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

### Train the model

```
model.fit(X_train, y_train, batch_size=512, validation_data=(X_test, y_test), epochs=3)
```

```
Epoch 1/3
50/50 [=====] - 915s 18s/step - loss: 0.5183 - accuracy: 0.7448 - val_loss: 0.4169 - val_accuracy: 0.8266
Epoch 2/3
50/50 [=====] - 816s 16s/step - loss: 0.3834 - accuracy: 0.8466 - val_loss: 0.3384 - val_accuracy: 0.8728
Epoch 3/3
50/50 [=====] - 830s 17s/step - loss: 0.3284 - accuracy: 0.8757 - val_loss: 0.2947 - val_accuracy: 0.8950
<keras.callbacks.History at 0x7f5f122bca90>
```

Make predictions using the BERT Model and classifying predictions values are within 0 and 1 (binary)

```

y_predictions = model.predict(X_test)
y_predictions = y_predictions.flatten()
y_predictions = np.where(y_predictions > 0.5, 1, 0)

```

## 4b. Performance Analysis of the Second Model (Pretrained BERT)

BERT uses a bidirectional representation of the input text, BERT takes into account both the left and right context of each word in the input sequence, while LSTMs only take into account the left context. This bidirectional representation allows BERT to capture more nuanced relationships between words in the input sequence. Therefore in the long run BERT Model is more superior than the first model. But the downside that BERT Model has a lot of parameters and takes a more time to train. Therefore for a specific/niche words to predict and train it is better to use a smaller model.

As for the results below are the metrics resulting also a good result within 512 batch size and 3 epochs. For a large parameters trained in BERT and considering the amount of dataset trained in BERT. A slightly lower accuracy of this doesn't mean that the first model is better, this results means that the BERT model is well fitted and the first model may be underfitted/overfitted.

```

print("f1-score: ", f1_score(y_test, y_predictions))
print("precision: ", precision_score(y_test, y_predictions))
print("recall: ", recall_score(y_test, y_predictions))
print("accuracy: ", accuracy_score(y_test, y_predictions))
print("\nClassification Report:\n", classification_report(y_test, y_predictions))

```

```

f1-score: 0.8987951807228916
precision: 0.8681990107652022
recall: 0.931626600624415
accuracy: 0.895

```

```

Classification Report:
              precision    recall  f1-score   support

     0       0.93         0.86         0.89         3197
     1       0.87         0.93         0.90         3203

   accuracy          0.90
  macro avg       0.90         0.89         0.89
 weighted avg     0.90         0.90         0.89

```