Hate classification Speech Using (Bert, Roberta, LSTM)

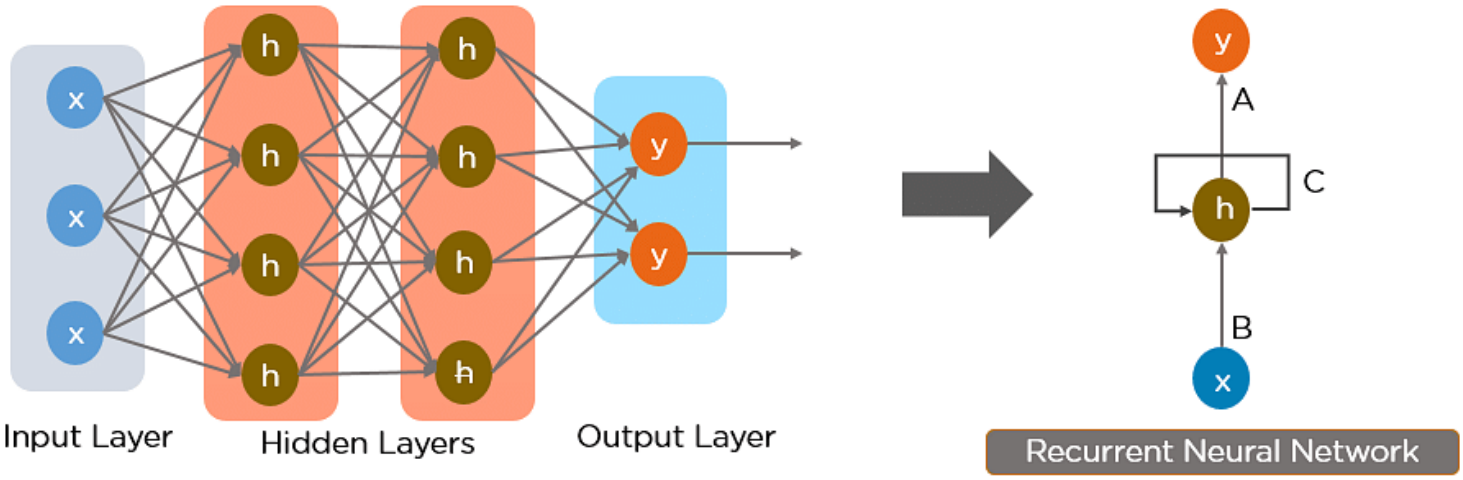
Team members:

Yomna Muhammed Alaa 2003125

Mannar Muhammed Saad 20201381434

Engy Muhammed Hassan 2003124

Shahd Ahmed Taher 20201447207



Our project is Hate speech classification where we try different models (LSTM as baseline, Bert and Roberta) but the process was long and different experiments were done to fine-tune and get the best results from Bert and compare the results between the three different models then in the final colab/notebook, we used the best fine tuned Bert to classify a text as hate or no hate.

#First data set: Hate Speech and Offensive Language Dataset

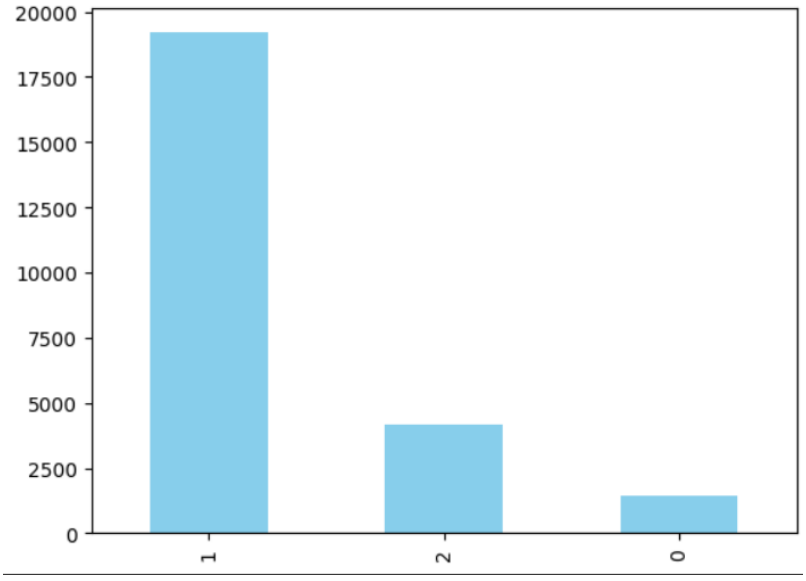
- **Dataset using Twitter data, is was used to research hate-speech detection. The text is classified as: hate-speech, offensive language, and neither. Due to the nature of the study, it's important to note that this dataset contains text that can be considered racist, sexist, homophobic, or generally offensive.**

I. After loading data we have to drop unused column to concentrate on needed ones

Dropped columns ['Unnamed: 0','count','hate_speech','offensive_language','neither']

II. This data has 3 classes {0 - hate speech 1 - offensive language 2 – neither}

```
1    19190
2     4163
0     1430
Name: class, dtype: int64
None
```



III. As we see in the previous step labels count has a huge different so to start using this data we have to decrease this difference

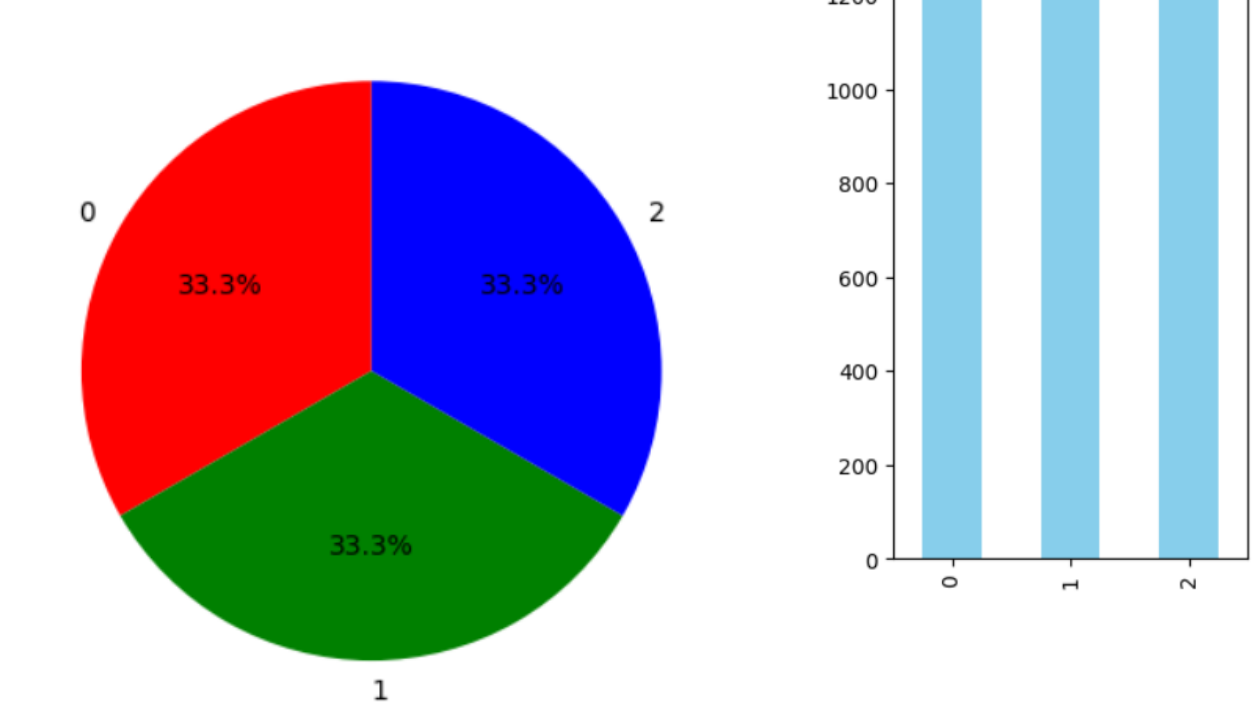
IV. Before starting we just has to make sure that there is not null values

V. To balance our data we used **UNDERSAMPLING:**

- ✓ is a technique to balance uneven datasets by keeping all of the data in the minority class and decreasing the size of the majority class.

VI. So the results that labels will be equal to the smaller one "1430"

```
Counter({0: 1430, 1: 1430, 2: 1430})
```



VII. Now our data is ready to get clean as we mentioned that we have tweet dataset which is not clean at all

- ❖ DATA CLEANING:

I. We found that the tweets contains username so we start getting ride of it by making func **remove_username_mentions**

II. Then we start removing punctuation, urlx , characters and number and lowercasing data with func **clean_tweet**

III. Then we did **Truecasing** {s the process of finding the proper capitalization of words in a text where the correct case information of the word is not given}

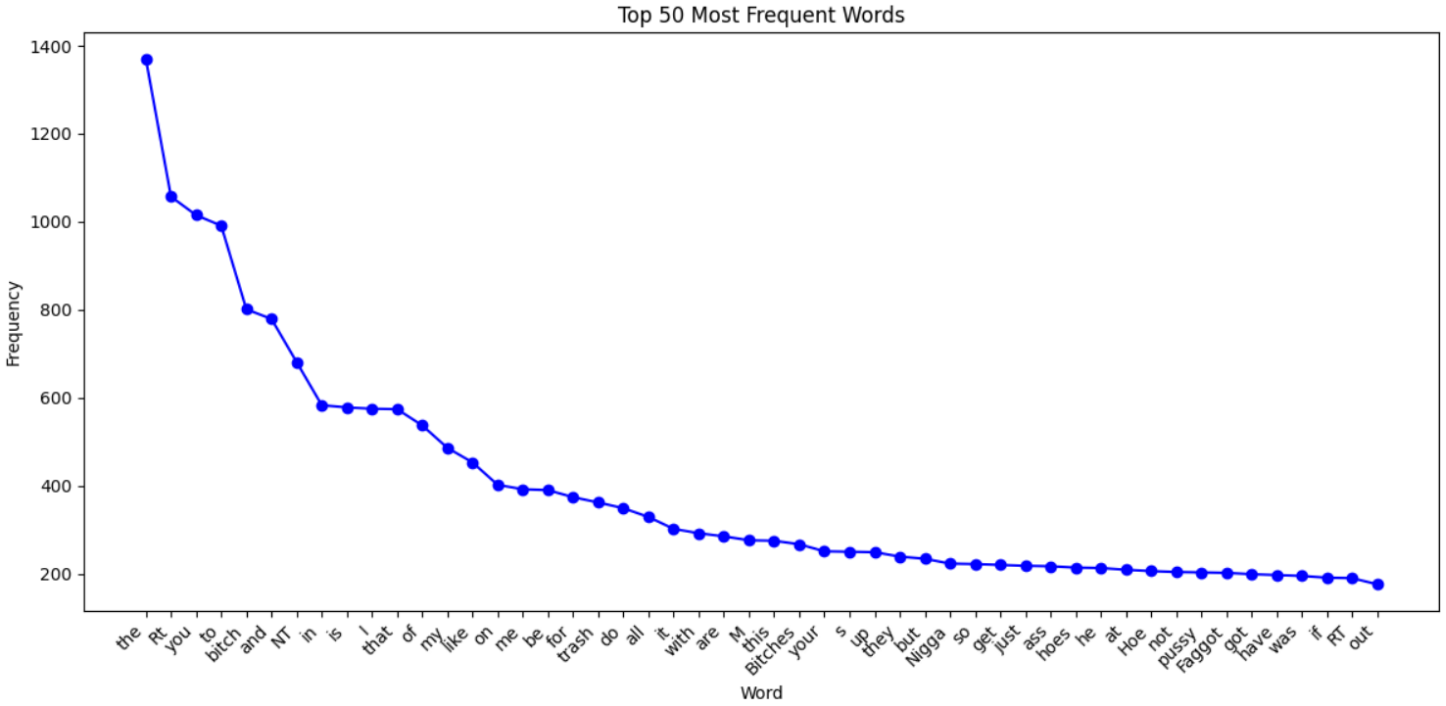
DATA BEFORE:

```
"@Blackman38Tide: @WhaleLookyHere @HowdyDowdy1...
"@CB_Baby24: @white_thunduh alsarabsss" hes a ...
"@DevilGrimz: @VigxRArts you're fucking gay, b...
"@MarkRoundtreeJr: LMFAOOOO I HATE BLACK PEOPL...
"@NoChillPaz: "At least I'm not a nigger" http...
                                                ...
RT @juliebenz: I drink unicorn tears... RT @th...
                  I see ya big Boi in the Oreo V's
```

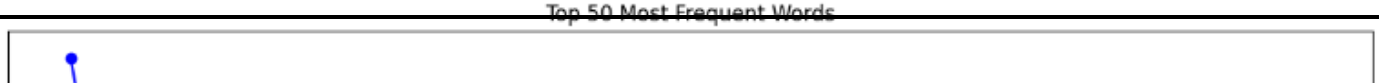
DATA AFTER:

```
Alsarabsss he s Beaner Smh you can tell he s M...
You re Fucking gay Blacklisted Hoe holding out...
Lmfaoooo hate black people this is why there s...
                    At least I m not nigger Lmfao
                                                ...
Rt drink Unicorn tears RT and they say blondes...
                   I see ya big Boi in the Oreo vs
```

AFTER TOKNIZATION:



AFTER REMOVING STOPING WORDS:



AFTER REMOVING STOPING WORDS:

```
                                  Queer Gaywad
              Alsarabsss Beaner Smh tell Mexican
    You Fucking gay Blacklisted Hoe holding anyway
   Lmfaoooo hate black people black people niggers
                          At least I nigger Lmfao
                        ...
  Rt drink Unicorn tears RT say blondes NT age w...
                         I see ya big Boi Oreo vs
  Rt Yankees four straight Shane Greene starts s...
                 Rt Rednecks fool around aux cord
  Did NT really one aware Yankees always loved D...
tweet_without_stopwords, Length: 4290, dtype: object
```

Word Cloud for training data without stopwords:

DATA SPLITING:

- We have split the data to Train and temp
- Then we split temp to test and validation

MODEL:

➢ BERT:

I. For preprocessing and encoder we used pretraining bert model from hugging face
II. Then we add the data to the neural network of 5 dense layers
III. Activation function softmax

```
Model: "model_12"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 text (InputLayer)           [(None,)]                 0         []

 keras_layer_4 (KerasLayer)  {'input_mask': (None, 128)  0         ['text[0][0]']
                             , 'input_word_ids': (None,
                              128),
                              'input_type_ids': (None,
                             128)}

 keras_layer_5 (KerasLayer)  {'pooled_output': (None, 7  109482241  ['keras_layer_4[0][0]',
                             68),                                   'keras_layer_4[0][1]',
                              'encoder_outputs': [(None              'keras_layer_4[0][2]']
                             , 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)],
                              'default': (None, 768),
                              'sequence_output': (None,
                              128, 768)}

 dropout (Dropout)           (None, 768)               0         ['keras_layer_5[0][13]']
```

OUTPUT RESULSTS:

```
Epoch 1/3
108/108 [==============================] - 2037s 19s/step - loss: 0.9174 - accuracy: 0.3333 - val_loss: 0.9015 - val_accuracy: 0.3372
Epoch 2/3
108/108 [==============================] - 1991s 18s/step - loss: 0.9171 - accuracy: 0.3333 - val_loss: 0.9012 - val_accuracy: 0.3372
Epoch 3/3
108/108 [==============================] - 1977s 18s/step - loss: 0.9170 - accuracy: 0.3333 - val_loss: 0.9012 - val_accuracy: 0.3372
```

➢ ROBERTA:

IV. For preprocessing and encoder we used pretraining Roberta model from hugging face

V. Then we add the data to the neural network of 5 dense layers

VI. Activation function Sigmoid

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

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

 keras_layer_1 (KerasLayer)     {'pooled_output': (  278042112   ['keras_layer[0][0]',
                                None, 768),                      'keras_layer[0][1]',
                                 'sequence_output':              'keras_layer[0][2]']
                                 (None, 128, 768),
                                 'encoder_outputs':
                                 [(None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
```

```
 dropout (Dropout)              (None, 768)          0          ['keras_layer_1[0][12]']

 dense (Dense)                  (None, 512)          393728     ['dropout[0][0]']

 dense_1 (Dense)                (None, 256)          131328     ['dense[0][0]']

 dense_2 (Dense)                (None, 128)          32896      ['dense_1[0][0]']

 dense_3 (Dense)                (None, 64)           8256       ['dense_2[0][0]']

 output (Dense)                 (None, 2)            130        ['dense_3[0][0]']

==================================================================================================
Total params: 278,608,450
Trainable params: 566,338
Non-trainable params: 278,042,112
__________________________________________________________________________________________________
```

OUTPUT RESULSTS:

```
Epoch 1/3
108/108 [==============================] - 1139s 10s/step - loss: 0.6921 - accuracy: 0.3365 - val_loss: 0.6638 - val_accuracy: 0.3372
Epoch 2/3
108/108 [==============================] - 1114s 10s/step - loss: 0.6671 - accuracy: 0.3333 - val_loss: 0.6634 - val_accuracy: 0.3372
Epoch 3/3
108/108 [==============================] - 1103s 10s/step - loss: 0.6669 - accuracy: 0.3333 - val_loss: 0.6631 - val_accuracy: 0.3372
```

➢ LSTM:

I. We have to add embedding layer to convert words into vectors
II. So first we have to get vocab size of training data
III. Then padding sentences to the same length
IV. Then LSTM layer
V. And last is dense layer with activation function sigmoid

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 409, 64)           1258112

 lstm (LSTM)                 (None, 64)                33024

 dense (Dense)               (None, 1)                 65

=================================================================
Total params: 1291201 (4.93 MB)
Trainable params: 1291201 (4.93 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

OUTPUT RESULSTS:

```
Epoch 1/10
3432/3432 - 56s - loss: -5.8575e+00 - accuracy: 0.3680 - 56s/epoch - 16ms/step
Epoch 2/10
```

So we tried another data set

Second data set: Dynamically Generated Hate Speech Dataset

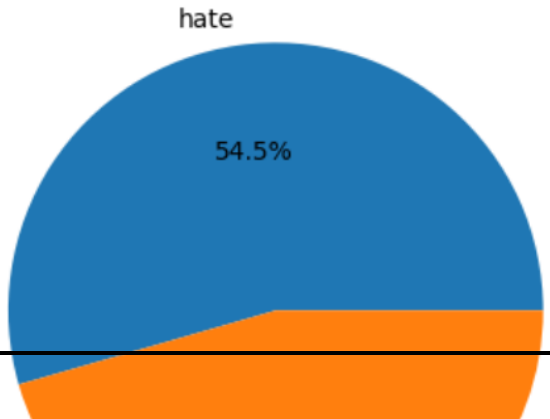
- dynamically Generated Datasets to Improve Online Hate Detection - A first-of-its-kind large synthetic training dataset for online hate classification, created from scratch with trained annotators over multiple rounds of dynamic data collection.

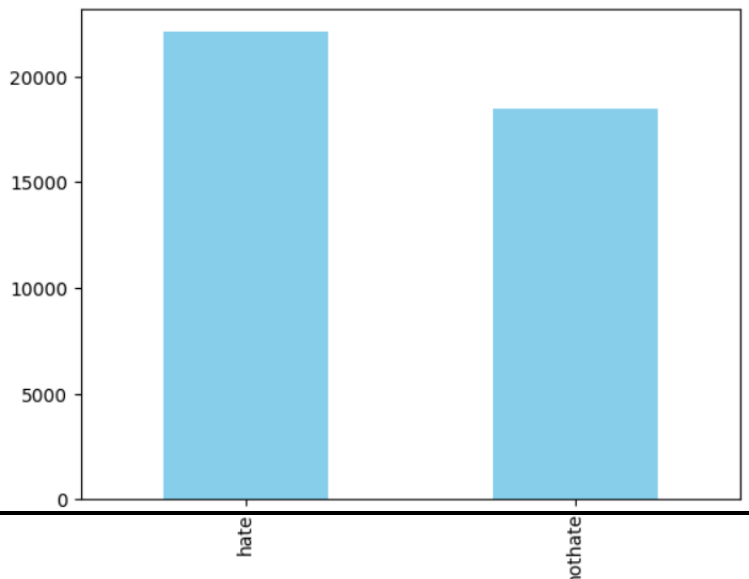
I. After loading data we have to drop unused column to concentrate on needed ones

Dropped columns:

['Unnamed:0','id','type','model_wrong','db.model_preds','status','round','split','annotator']

II. This data has 2 classes {hate ,nothate}





I. As we see in the previous step labels count has just small difference in data set

II. Before starting we just has to make sure that there is not null values

```
Total Null values count:  text     0
label    0
dtype: int64
```

III. We noticed that this data has less problems in cleaning

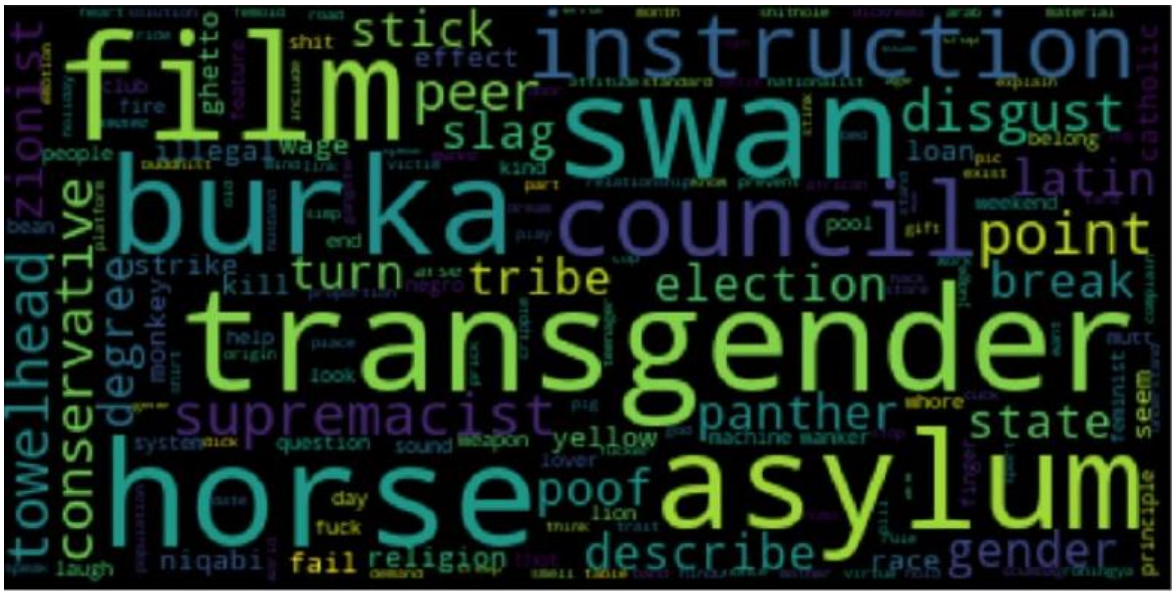
- ✓ First: we have to lowercasing the data
- ✓ Second: we noticed that the data has a lot of contractions **expand_contractions** to solve this problem
- ✓ Third : we removed extra spaces and punctuations

Before cleaning:

```
text 1:
 pear is not tasty
text 2:
 blonde ladies are fuc''ing smart
text 3:
 blonde people are not smart
text 4:
 Arabs are terrorists
text 5:
 Terrorists must be Muslims
text 6:
 Terrorists must be Muslim rats
```

After cleaning :

```
text 1:
 pear is not tasty
text 2:
 blonde ladies are fucing smart
text 3:
 blonde people are not smart
text 4:
 Arabs are terrorists
text 5:
 Terrorists must be Muslims
text 6:
```



DATA SPLITING:

- We have split the data to Train and temp
- Then we split temp to test and validation

MODEL:

➢ BERT:

I. epochs=3 OutPut:

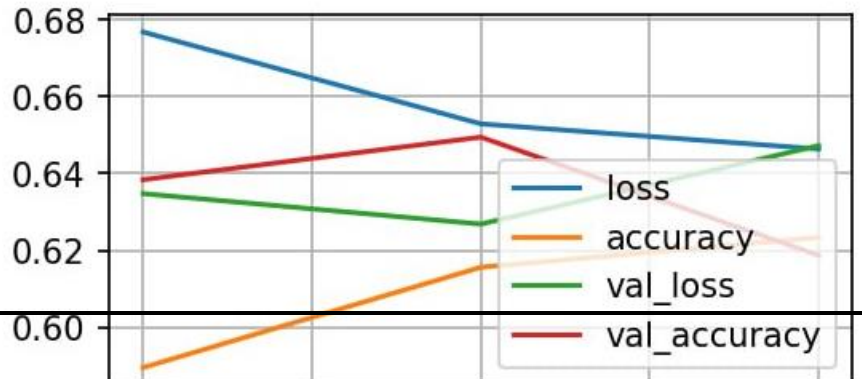
```
[ ] history = model.fit(X_train, y_train, validation_data= [X_valid,y_valid],epochs=3)

    Epoch 1/3
    1016/1016 [==============================] - 377s 347ms/step - loss: 0.6768 - accuracy: 0.5895 - val_loss: 0.6348 - val_accuracy: 0.6384
    Epoch 2/3
    1016/1016 [==============================] - 355s 350ms/step - loss: 0.6529 - accuracy: 0.6156 - val_loss: 0.6268 - val_accuracy: 0.6494
    Epoch 3/3
    1016/1016 [==============================] - 354s 348ms/step - loss: 0.6464 - accuracy: 0.6233 - val_loss: 0.6472 - val_accuracy: 0.6187
```

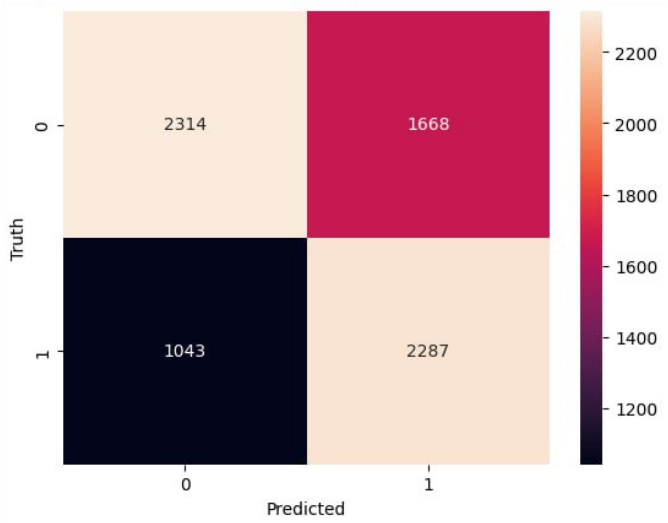
evaluation:

```
229/229 [==============================] - 79s 343ms/step - loss: 0.6493 - accuracy: 0.6183
[0.649296760559082, 0.6182987093925476]
```

plots:



Confusion Matrix:



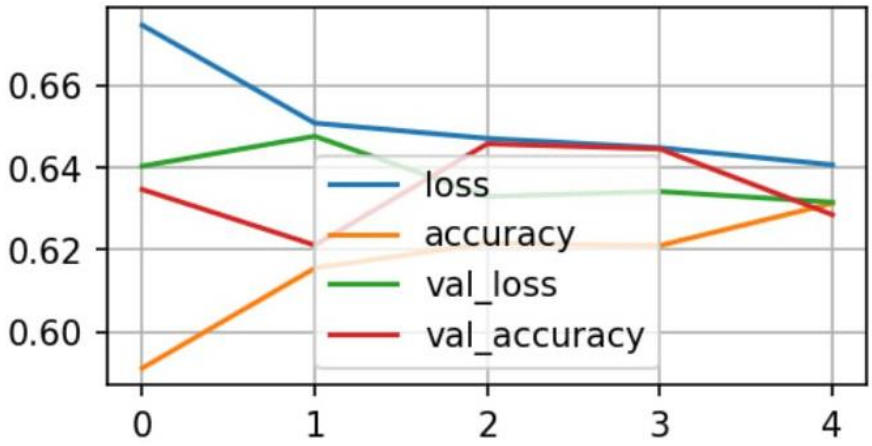
II. epochs=8 OutPut:

```
Epoch 1/5
1016/1016 [==============================] - 379s 358ms/step - loss: 0.6746 - accuracy: 0.5913 - val_loss: 0.6403 - val_accuracy: 0.6347
Epoch 2/5
1016/1016 [==============================] - 360s 354ms/step - loss: 0.6508 - accuracy: 0.6155 - val_loss: 0.6476 - val_accuracy: 0.6212
Epoch 3/5
1016/1016 [==============================] - 361s 356ms/step - loss: 0.6471 - accuracy: 0.6217 - val_loss: 0.6330 - val_accuracy: 0.6458
Epoch 4/5
1016/1016 [==============================] - 361s 355ms/step - loss: 0.6448 - accuracy: 0.6211 - val_loss: 0.6342 - val_accuracy: 0.6445
Epoch 5/5
1016/1016 [==============================] - 361s 355ms/step - loss: 0.6407 - accuracy: 0.6313 - val_loss: 0.6316 - val_accuracy: 0.6285
```

Evaluation :

```
229/229 [==============================] - 78s 342ms/step - loss: 0.6284 - accuracy: 0.6398
[0.6283723711967468, 0.6397702693939209]
```

Plots:



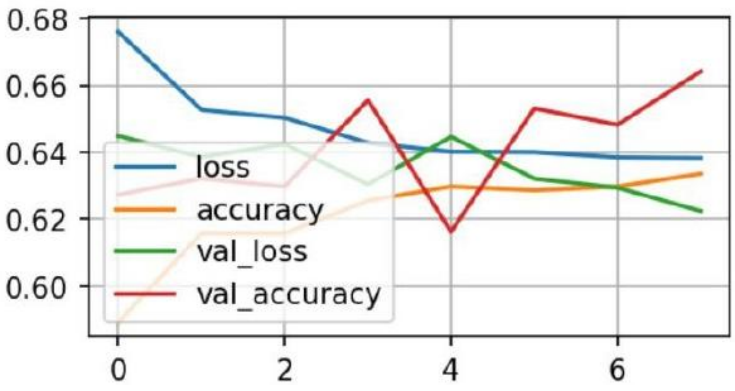
III. epochs=8 OutPut:

```
Epoch 1/8
1016/1016 [==============================] - 382s 358ms/step - loss: 0.6760 - accuracy: 0.5892 - val_loss: 0.6449 - val_accuracy: 0.6273
Epoch 2/8
1016/1016 [==============================] - 363s 357ms/step - loss: 0.6527 - accuracy: 0.6158 - val_loss: 0.6386 - val_accuracy: 0.6322
Epoch 3/8
1016/1016 [==============================] - 363s 357ms/step - loss: 0.6503 - accuracy: 0.6157 - val_loss: 0.6425 - val_accuracy: 0.6298
Epoch 4/8
1016/1016 [==============================] - 362s 356ms/step - loss: 0.6428 - accuracy: 0.6255 - val_loss: 0.6304 - val_accuracy: 0.6556
Epoch 5/8
1016/1016 [==============================] - 361s 355ms/step - loss: 0.6402 - accuracy: 0.6298 - val_loss: 0.6446 - val_accuracy: 0.6162
Epoch 6/8
1016/1016 [==============================] - 362s 356ms/step - loss: 0.6401 - accuracy: 0.6286 - val_loss: 0.6321 - val_accuracy: 0.6531
Epoch 7/8
1016/1016 [==============================] - 362s 356ms/step - loss: 0.6385 - accuracy: 0.6298 - val_loss: 0.6294 - val_accuracy: 0.6482
Epoch 8/8
1016/1016 [==============================] - 361s 355ms/step - loss: 0.6383 - accuracy: 0.6336 - val_loss: 0.6224 - val_accuracy: 0.6642
```

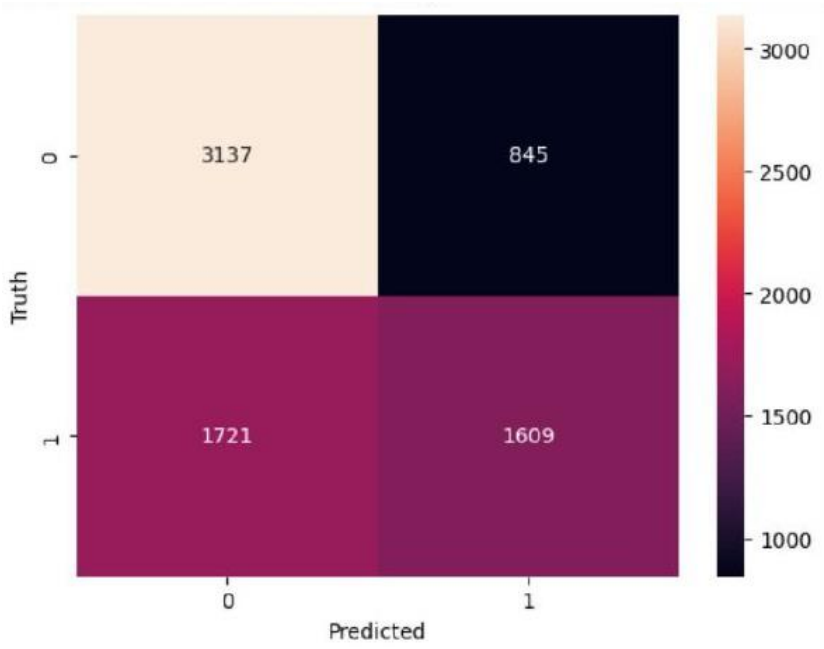
Evalution:

```
229/229 [==============================] - 79s 345ms/step - loss: 0.6222 - accuracy: 0.6504
[0.6221644878387451, 0.6504376530647278]
```

Plotting:



Confusion matrix:



➢ LSTM:

I. 10 epochs

```
Epoch 1/10
32498/32498 - 482s - loss: 0.5696 - accuracy: 0.6827 - 482s/epoch - 15ms/step
Epoch 2/10
32498/32498 - 437s - loss: 0.4139 - accuracy: 0.7993 - 437s/epoch - 13ms/step
Epoch 3/10
32498/32498 - 426s - loss: 0.3118 - accuracy: 0.8617 - 426s/epoch - 13ms/step
Epoch 4/10
32498/32498 - 424s - loss: 0.2391 - accuracy: 0.8975 - 424s/epoch - 13ms/step
Epoch 5/10
32498/32498 - 448s - loss: 0.1863 - accuracy: 0.9241 - 448s/epoch - 14ms/step
Epoch 6/10
32498/32498 - 431s - loss: 0.1453 - accuracy: 0.9425 - 431s/epoch - 13ms/step
Epoch 7/10
32498/32498 - 439s - loss: 0.1155 - accuracy: 0.9548 - 439s/epoch - 14ms/step
Epoch 8/10
32498/32498 - 435s - loss: 0.0929 - accuracy: 0.9645 - 435s/epoch - 13ms/step
Epoch 9/10
32498/32498 - 433s - loss: 0.0748 - accuracy: 0.9721 - 433s/epoch - 13ms/step
Epoch 10/10
32498/32498 - 432s - loss: 0.0621 - accuracy: 0.9773 - 432s/epoch - 13ms/step
<keras.src.callbacks.History at 0x7b8d12058430>
```