PHASE 2

SENTIMENTAL ANALYSIS FOR MARKETING

MEMBERS

- SOUNDHAR BALAJI.B
- RADHIKA.M
- ROHANSHAJ.K.R
- VIGNESH.V
- NELSON JOSEPH.M

Part I INTRODUCTION

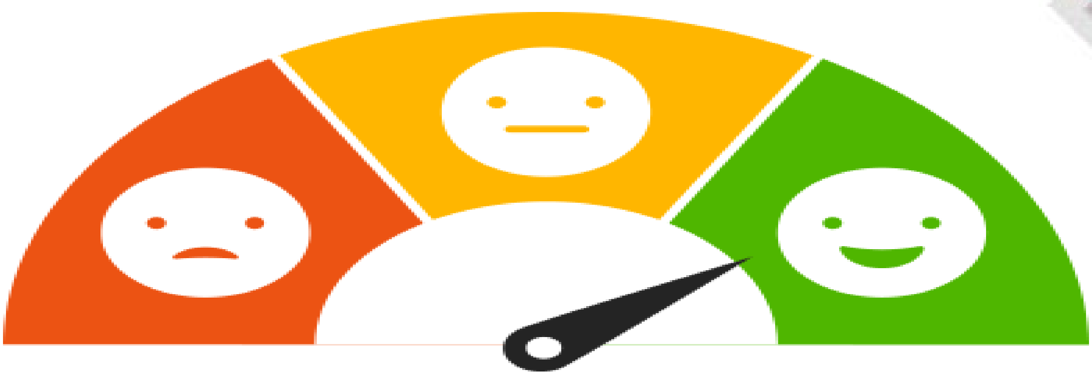
In the realm of marketing, sentiment analysis plays a pivotal role in understanding how customers perceive products, brands, and services. It enables marketers to extract valuable insights, uncover trends, and make data-driven decisions that can impact market

In the realm of marketing, sentiment analysis plays a pivotal role in understanding how customers perceive products, brands, and services. It enables marketers to extract valuable insights, uncover trends, and make data-driven decisions that can impact marketing strategies, product development, and customer satisfaction.

In this age of information overload, sentiment analysis is an invaluable tool that empowers marketers to make informed decisions based on the collective voice of their customers and target audience. By harnessing the power of NLP and machine learning, businesses can gain a competitive edge, enhance customer experiences, and drive more effective marketing campaigns.

PHASE 2 Innovation :

Explore advanced techniques like fine-tuning pre-trained sentiment analysis models (BERT, RoBERTa) for more accurate sentiment predictions.



How Does Sentiment Analysis Work?

1. **Text Preprocessing:** The text data is cleaned by removing irrelevant information, such as special characters, punctuation, and stopwords.
2. **Tokenization:** The text is divided into individual words or tokens to facilitate analysis.
3. **Feature Extraction:** Relevant features are extracted from the text, such as words, n-grams, or even parts of spee
4. **Sentiment Classification:** Machine learning algorithms or pre-trained models are used to classify the sentiment of each text instance. This can be achieved through supervised learning, where models are trained on labeled data, or through pre-trained models that have learned sentiment patterns from large datasets.
5. **Post-processing:** The sentiment analysis results may undergo additional processing, such as aggregating sentiment scores or applying threshold rules to classify sentiments as positive, negative, or neutral.
6. **Evaluation:** The performance of the sentiment analysis model is assessed using evaluation metrics, such as accuracy, precision, recall, or F1 score.

MODELS USED :

1. BERT
https://colab.research.google.com/github/Rutu07/Sentiment-Analysis-Using-BERT/blob/main/Airline_Tweets_Sentiment_Analysis_Using_BERT(1).ipynb#scrollTo=KrPatVOO0GQX
2.ROBERTA
https://colab.research.google.com/github/DhavalTaunk08/NLP_scripts/blob/master/sentiment_analysis_using_roberta.ipynb

DATASET : https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment/data

Part II PROGRAM - BERT

```
import torch
import pandas as pd
import numpy as np
import re
from sklearn.model_selection import train_test_split
pd.set_option('display.max_colwidth',200)
from sklearn.preprocessing import LabelEncoder

import matplotlib.pyplot as plt

#library for progress bar
from tqdm import notebook
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler

# importing nn module
import torch.nn as nn

#library for computing class weights
from sklearn.utils.class_weight import compute_class_weight

from sklearn.metrics import classification_report
import time
import datetime
```

```
# Checking if GPU is available.
if torch.cuda.is_available():
    device=torch.device('cuda')
```

```
print(device)
torch.cuda.get_device_name(0)
# Current GPU is Tesla T4
```

```
:# Step 2: Installing Hugging Face's Transformers Library
# Hugging face is one of the most popular NLP library and provides a wide range of transformer-based models such as BERT, GPT-2, Roberta, and so on.

!pip install transformers
```

```
# Step 3: Installing BertModel
from transformers.models.bert.modeling_bert import BertModel

# Import BERT pretrained module
from transformers import BertModel
#Download uncased bert base model
bert=BertModel.from_pretrained('bert-base-uncased')
```

```
# Print BERT arcitecture
print(bert)
```

OUTPUT

```
# Print BERT arcitecture
print(bert)
account_circle
BertModel(
  (embeddings): BertEmbeddings(
    (word_embeddings): Embedding(30522, 768, padding_idx=0)
    (position_embeddings): Embedding(512, 768)
    (token_type_embeddings): Embedding(2, 768)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (encoder): BertEncoder(
    (layer): ModuleList(
      (0): BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
          (dense): Linear(in_features=768, out_features=3072, bias=True)
```

```
        (intermediate_act_fn): GELUActivation()
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (1): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
        (intermediate_act_fn): GELUActivation()
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (2): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
```

```
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
        (intermediate_act_fn): GELUActivation()
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (3): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
        (intermediate_act_fn): GELUActivation()
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (4): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
```

```
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
    (intermediate_act_fn): GELUActivation()
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(5): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
    (intermediate_act_fn): GELUActivation()
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(6): BertLayer(
```

```
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
        (intermediate_act_fn): GELUActivation()
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (7): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
        (intermediate_act_fn): GELUActivation()
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
```

```
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (8): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
      (intermediate_act_fn): GELUActivation()
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (9): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
```

```
      (dense): Linear(in_features=768, out_features=3072, bias=True)
      (intermediate_act_fn): GELUActivation()
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (10): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
      (intermediate_act_fn): GELUActivation()
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (11): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
```

```
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
        (intermediate_act_fn): GELUActivation()
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
  )
)
(pooler): BertPooler(
  (dense): Linear(in_features=768, out_features=768, bias=True)
  (activation): Tanh()
)
)
```

```
#Step 4: Importing BERT tokenizer

from transformers.models.bert.tokenization_bert_fast import BertTokenizerFast

# importing BERT tokenizer tokenizer=BertTokenizerFast.from_pretrained('bert-base-uncased',do_lower_case=True)
```

```
# converting integers back to text
print("Tokenizer Text: ",tokenizer.convert_ids_to_tokens(sentence_id))
```

```
text='Jim Henson was a puppeteer'
sentence_id=tokenizer.encode(text,
                        # add special character tokens
                        add_special_tokens=True,
                        # Specifying maximum length for any input sequences
                        max_length=10,
                        # if exceeeding 10, then it will be truncated, if <10, then it will be padded.
                        truncation=True,
                        # add pad tokens to the right side of the sequence
                        pad_to_max_length='right'
                        )
print("Integer Sequence:{}".format(sentence_id))
```

OUTPUT:

Tokenizer Text: ['[CLS]', 'jim', 'henson', 'was', 'a', 'puppet', '##eer', '[SEP]', '[PAD]', '[PAD]']

```
decoded=tokenizer.decode(sentence_id)

# Print BERT arcitecture
```

OUTPUT:

Decoded String:[CLS] jim henson was a puppeteer [SEP] [PAD] [PAD]

```
att_mask=[int(tok>0) for tok in sentence_id]
print(att_mask)
```

OUTPUT:

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

```
# convert lists to tensors
# torch.tensor creates a tensor of given data
sent_id=torch.tensor(sentence_id)
attn_mask=torch.tensor(att_mask)
print('Shape of sentence_id before reshaping is: {}'.format(sent_id.shape))
print('Shape of sentence_id before reshaping is: {}'.format(attn_mask.shape))
print('\n')
# reshaping tensor in form of batch,text length
sent_id=sent_id.unsqueeze(0)
attn_mask=attn_mask.unsqueeze(0)
print('Shape of sentence_id after reshaping is: {}'.format(sent_id.shape))
print('Shape of sentence_id after reshaping is: {}'.format(attn_mask.shape))
print('\n')
# reshaped tensor
print(sent_id)
```

OUTPUT:

Shape of sentence_id before reshaping is: torch.Size([10])
Shape of sentence_id before reshaping is: torch.Size([10])
Shape of sentence_id after reshaping is: torch.Size([1, 10])
Shape of sentence_id after reshaping is: torch.Size([1, 10])
tensor([[101, 3958, 27227, 2001, 1037, 13997, 11510, 102, 0,
0]])

```
# passing integer sequence and attention mask tensor to BERT model
outputs=bert(sent_id,attention_mask=attn_mask)
```

```
# Unpacking the output of BERT model

# all_hidden_states is a collection of all the output vectors/ hidden states (of encoder) at each timestamps or position of the BERT model
all_hidden_states=outputs[0]
```

OUTPUT:

```
torch.Size([1, 10, 768])
tensor([[[-0.2531,  0.2038, -0.3862,  ..., -0.3034,  0.6197,  0.2373],
         [-0.2323, -0.0044, -0.5479,  ...,  0.0765,  0.8122, -0.4710],
         [ 0.2590,  0.7140, -0.5438,  ..., -0.3774,  0.9987,  0.5400],
         ...,
         [ 0.7873,  0.3299, -0.0351,  ...,  0.2932, -0.5141,  0.0308],
         [-0.5547, -0.3669, -0.1106,  ...,  0.2593,  0.5321, -0.3871],
         [-0.5461, -0.2414, -0.2111,  ...,  0.3100,  0.5863, -0.3467]]],
       grad_fn=<NativeLayerNormBackward0>)
```

```
# this output contains output vector against the CLS token only (at the first position of BERT model)
# this output vector encodes the entire input sequence

cls_hidden_state=outputs[1]

print(cls_hidden_state.shape)
print(cls_hidden_state)
```

OUTPUT:

```
torch.Size([1, 768])
tensor([[-0.8767, -0.4109, -0.1220,  0.4494,  0.1945, -0.2698,  0.8316,  0.3127,
          0.1178, -1.0000, -0.1561,  0.6677,  0.9891, -0.3451,  0.8812, -0.6753,
         -0.3079, -0.5580,  0.4380, -0.4588,  0.5831,  0.9956,  0.4467,  0.2863,
          0.3924,  0.6864, -0.7513,  0.9043,  0.9436,  0.8207, -0.6493,  0.3524,
         -0.9919, -0.2295, -0.0742, -0.9936,  0.3698, -0.7558,  0.0792, -0.2218,
         -0.8637,  0.4711,  0.9997, -0.4368,  0.0404, -0.3498, -1.0000,  0.2663,
         -0.8711,  0.0508,  0.0505, -0.1634,  0.1716,  0.4363,  0.4330, -0.0333,
         -0.0416,  0.2206, -0.2568, -0.6122, -0.5916,  0.2569, -0.2622, -0.9041,
          0.3221, -0.2394, -0.2634, -0.3454, -0.0723,  0.0081,  0.8297,  0.2279,
          0.1614, -0.6555, -0.2062,  0.3280, -0.4016,  1.0000, -0.0952, -0.9874,
         -0.0400,  0.0717,  0.3675,  0.3373, -0.3710, -1.0000,  0.4479, -0.1722,
         -0.9917,  0.2677,  0.4844, -0.2207, -0.3207,  0.3715, -0.2171, -0.2522,
         -0.3071, -0.3161, -0.1988, -0.0860, -0.0114, -0.1982, -0.1799, -0.3221,
          0.1751, -0.4442, -0.1570, -0.0434, -0.0893,  0.5717,  0.3112, -0.2900,
          0.3305, -0.9430,  0.6061, -0.2984, -0.9873, -0.3956, -0.9926,  0.7857,
```

-0.1692, -0.2719, 0.9505, 0.5628, 0.2904, -0.1693, 0.1619, -1.0000,
-0.1697, -0.1534, 0.2513, -0.2857, -0.9846, -0.9638, 0.5565, 0.9200,
0.1805, 0.9995, -0.2122, 0.9391, 0.3246, -0.3937, -0.1248, -0.5209,
0.0519, 0.1141, -0.6463, 0.3529, -0.0322, -0.3837, -0.3796, -0.2830,
0.1280, -0.9191, -0.4201, 0.9145, 0.0713, -0.2455, 0.5212, -0.2642,
-0.3675, 0.8082, 0.2577, 0.2755, -0.0157, 0.3675, -0.3107, 0.4502,
-0.8224, 0.2841, 0.4360, -0.3193, 0.2164, -0.9851, -0.4444, 0.5759,
0.9878, 0.7531, 0.3384, 0.2003, -0.2602, 0.4695, -0.9561, 0.9855,
-0.1712, 0.2295, 0.1220, -0.1386, -0.8436, -0.3783, 0.8371, -0.3204,
-0.8457, -0.0473, -0.4219, -0.3593, -0.2187, 0.5282, -0.3149, -0.4375,
-0.0440, 0.9242, 0.9296, 0.7735, -0.3733, 0.3945, -0.9049, -0.2898,
0.2695, 0.2910, 0.1695, 0.9932, -0.3069, -0.1611, -0.8349, -0.9827,
0.1299, -0.8555, -0.0531, -0.6830, 0.3926, 0.2873, -0.1899, 0.2598,
-0.9201, -0.7455, 0.3943, -0.3955, 0.4015, -0.2341, 0.7593, 0.3421,
-0.6143, 0.5170, 0.8987, 0.1072, -0.6858, 0.6481, -0.2454, 0.8712,
-0.5958, 0.9936, 0.3404, 0.4972, -0.9452, -0.2347, -0.8748, -0.0154,
-0.1293, -0.5265, 0.4235, 0.4206, 0.3663, 0.7488, -0.4650, 0.9900,
-0.8695, -0.9701, -0.5203, -0.0900, -0.9914, 0.0978, 0.2844, -0.0424,
-0.4649, -0.4546, -0.9620, 0.8035, 0.2177, 0.9705, -0.0793, -0.7985,
-0.3436, -0.9537, -0.0035, -0.0945, 0.4291, 0.0391, -0.9602, 0.4497,
0.5135, 0.4913, 0.0608, 0.9948, 1.0000, 0.9810, 0.8865, 0.7961,
-0.9894, -0.5122, 1.0000, -0.8521, -1.0000, -0.9412, -0.6633, 0.3110,
-1.0000, -0.1468, -0.1235, -0.9465, -0.0891, 0.9796, 0.9700, -1.0000,
0.9324, 0.9259, -0.4503, 0.4591, -0.1785, 0.9819, 0.2285, 0.4423,
-0.2615, 0.4124, -0.5252, -0.8534, 0.0365, -0.0670, 0.8944, 0.1913,
-0.4782, -0.9402, 0.2293, -0.1581, -0.2440, -0.9604, -0.1924, -0.0555,
0.5484, 0.1915, 0.2038, -0.7367, 0.2698, -0.7307, 0.3715, 0.5640,
-0.9386, -0.5717, 0.3818, -0.2775, 0.1536, -0.9608, 0.9702, -0.3502,
0.1524, 1.0000, 0.3876, -0.9001, 0.2547, 0.1857, 0.0832, 1.0000,
0.3811, -0.9852, -0.4053, 0.2576, -0.3923, -0.4125, 0.9994, -0.1463,
-0.0428, 0.2818, 0.9899, -0.9923, 0.8351, -0.8563, -0.9634, 0.9617,
0.9268, -0.4225, -0.7369, 0.1318, 0.1107, 0.2294, -0.8914, 0.6082,
0.4665, -0.0720, 0.8555, -0.7973, -0.3478, 0.4201, -0.1762, 0.0761,
0.2823, 0.4571, -0.1350, 0.1190, -0.3509, -0.4039, -0.9556, 0.0262,
1.0000, -0.2164, 0.0569, -0.2296, -0.1003, -0.1827, 0.4036, 0.4715,
-0.3293, -0.8471, -0.0518, -0.8453, -0.9935, 0.6732, 0.2284, -0.1968,
0.9998, 0.5194, 0.2326, 0.1718, 0.7497, -0.0192, 0.4518, -0.0327,
0.9765, -0.3259, 0.3491, 0.7471, -0.3186, -0.3019, -0.5725, 0.0563,
-0.9206, 0.0572, -0.9589, 0.9565, 0.3109, 0.3348, 0.1635, -0.0619,
1.0000, -0.6020, 0.5309, -0.3723, 0.6636, -0.9851, -0.6789, -0.4312,
-0.1435, -0.0827, -0.2497, 0.1323, -0.9786, -0.0474, -0.0304, -0.9444,
-0.9927, 0.2508, 0.6172, 0.1679, -0.7980, -0.6078, -0.4906, 0.4646,
-0.1934, -0.9396, 0.5453, -0.3000, 0.4329, -0.3340, 0.4408, -0.2058,
0.8344, 0.1265, -0.0307, -0.2098, -0.8340, 0.7114, -0.7410, 0.0518,

```
-0.1481,    1.0000, -0.3100,    0.1461,    0.7011,    0.6334, -0.2857,    0.1618,
  0.0966,    0.2955, -0.0981, -0.1832, -0.6208, -0.3013,    0.4337,    0.0283,
-0.2959,    0.7579,    0.4711,    0.3666, -0.0531,    0.0914,    0.9969, -0.2267,
-0.1165, -0.5533, -0.1262, -0.3575, -0.2124,    1.0000,    0.3679,    0.0604,
-0.9936, -0.2000, -0.9208,    0.9999,    0.8511, -0.8783,    0.5650,    0.2405,
-0.2859,    0.6935, -0.2598, -0.2655,    0.2893,    0.2862,    0.9774, -0.4575,
-0.9764, -0.5964,    0.3966, -0.9575,    0.9939, -0.5326, -0.2349, -0.4376,
-0.0250,    0.2574,    0.0274, -0.9762, -0.1582,    0.1821,    0.9811,    0.3014,
-0.3820, -0.9007, -0.1151,    0.3936, -0.0680, -0.9449,    0.9809, -0.9313,
  0.2600,    1.0000,    0.3860, -0.5243,    0.2401, -0.4410,    0.3253, -0.1413,
  0.5428, -0.9466, -0.2817, -0.3262,    0.4330, -0.2120, -0.2457,    0.7247,
  0.2134, -0.3430, -0.6305, -0.1214,    0.4871,    0.7498, -0.2957, -0.1829,
  0.1699, -0.1391, -0.9264, -0.4167, -0.2995, -0.9991,    0.6411, -1.0000,
-0.1510, -0.5473, -0.2219,    0.8075,    0.3862, -0.1392, -0.7206, -0.0710,
  0.6995,    0.6656, -0.2889,    0.2902, -0.6951,    0.1622, -0.1298,    0.3182,
  0.1694,    0.6526, -0.2735,    1.0000,    0.1370, -0.3043, -0.9189,    0.3041,
-0.2604,    1.0000, -0.7969, -0.9715,    0.2110, -0.5773, -0.7218,    0.2477,
-0.0304, -0.7015, -0.6577,    0.9111,    0.8219, -0.3693,    0.4537, -0.3062,
-0.3671,    0.0856,    0.1595,    0.9903,    0.2790,    0.8213, -0.2885, -0.0724,
  0.9636,    0.2213,    0.6892,    0.2070,    1.0000,    0.3249, -0.8999,    0.2644,
-0.9700, -0.2610, -0.9228,    0.4016,    0.1170,    0.8570, -0.3587,    0.9672,
  0.0667,    0.1108, -0.1840,    0.4711,    0.3127, -0.9391, -0.9892, -0.9908,
  0.3962, -0.5013, -0.0640,    0.3811,    0.1530,    0.4712,    0.3781, -1.0000,
  0.9466,    0.3529,    0.2077,    0.9735,    0.2019,    0.4726,    0.4248, -0.9892,
-0.9203, -0.3418, -0.2910,    0.6572,    0.5584,    0.8190,    0.4319, -0.4171,
-0.4697,    0.4653, -0.8583, -0.9940,    0.4802,    0.0740, -0.8986,    0.9559,
-0.4745, -0.1616,    0.4457,    0.1412,    0.8933,    0.8280,    0.4313,    0.2437,
  0.6787,    0.9043,    0.8940,    0.9903, -0.2561,    0.6986, -0.0055,    0.3281,
  0.6809, -0.9586,    0.1583,    0.0033, -0.2711,    0.3025, -0.1928, -0.9207,
  0.5260, -0.2139,    0.5709, -0.2302,    0.1593, -0.4779, -0.1577, -0.7036,
-0.5208,    0.4676,    0.2335,    0.9372,    0.4775, -0.1995, -0.5655, -0.2336,
  0.0798, -0.9315,    0.8288, -0.0946,    0.5294,    0.0223, -0.0744,    0.7821,
  0.1236, -0.3705, -0.3959, -0.7528,    0.8145, -0.3204, -0.4786, -0.5135,
  0.7306,    0.3208,    0.9981, -0.3959, -0.3492, -0.1118, -0.2872,    0.3596,
-0.1345, -1.0000,    0.2896,    0.2262,    0.1702, -0.3530,    0.1111, -0.0755,
-0.9565, -0.2658,    0.2530, -0.0490, -0.5834, -0.4616,    0.3937,    0.2329,
  0.5620,    0.8138, -0.0288,    0.5621,    0.3811,    0.0852, -0.6049,    0.8452]],
grad_fn=<TanhBackward0>)
```

```
#Step 5: Data Preparation


!unzip 'Airline_Tweets-200904-165552.zip'
df=pd.read_csv('Tweets.csv')
df.head()
```

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline	airline_sentiment_gold	name	negativereason_gold	retweet_count	text	tweet_coord	tweet_created	tweet_location	user_timezone
0	570306133677760513	neutral	1.0000	NaN	NaN	Virgin America	NaN	cairdin	NaN	0	@VirginAmerica What @dhepburn said.	NaN	2015-02-24 11:35:52 -0800	NaN	Eastern Time (US & Canada)
1	570301130888122368	positive	0.3486	NaN	0.0000	Virgin America	NaN	jnardino	NaN	0	@VirginAmerica plus you've added commercials to the experience... tacky.	NaN	2015-02-24 11:15:59 -0800	NaN	Pacific Time (US & Canada)
2	570301083672813571	neutral	0.6837	NaN	NaN	Virgin America	NaN	yvonnalynn	NaN	0	@VirginAmerica I didn't today... Must mean I need to take another trip!	NaN	2015-02-24 11:15:48 -0800	Lets Play	Central Time (US & Canada)
3	570301031407624196	negative	1.0000	Bad Flight	0.7033	Virgin America	NaN	jnardino	NaN	0	@VirginAmerica it's really aggressive to blast obnoxious "entertainment" in your guests' faces & they have little recourse	NaN	2015-02-24 11:15:36 -0800	NaN	Pacific Time (US & Canada)
4	570300817074462722	negative	1.0000	Can't Tell	1.0000	Virgin America	NaN	jnardino	NaN	0	@VirginAmerica and it's a really big bad thing about it	NaN	2015-02-24 11:14:45 -0800	NaN	Pacific Time (US & Canada)

```
df.shape
```

OUTPUT:

6520 @SouthwestAir You officially have the worst customer service of any airline I've ever dealt with. #southwestairlines #poor
8851 @JetBlue I know where you guys jet! LOL, but if you love me so much, help a brother out :) Hot weather, great nightlife, 2-3 hour flight
6927 @JetBlue flight for tomorrow morning Cancelled Flighted & can't seem to rebook me. They can't even get me a seat. No clear answer on why Cancelled Flighted.
13263
@AmericanAir SJC->LAX. After the fourth time, I gave up!
3587
@united & I've been hung up on twice by your staff. So upset right now
Name: text, dtype: object

```
print(df['airline_sentiment'].value_counts())
print(df['airline_sentiment'].value_counts(normalize=True))
```

OUTPUT:

negative 9178
neutral 3099
positive 2363
Name: airline_sentiment, dtype: int64
negative 0.626913
neutral 0.211680
positive 0.161407
Name: airline_sentiment, dtype: float64

```
# Sabing value counts to a list
class_counts=df['airline_sentiment'].value_counts().to_list()
```

```
# using apply function to apply this preprocess function on each row of the text column
df['cleaned_text']=df['text'].apply(preprocess)
```

```
    text=re.sub(r'@[A-Za-z0-9]+','',text)
    text=re.sub(r'http\S+','',text)
    tokens=text.split()
return ' '.join(tokens)
```

```
df.head()[['airline_sentiment','text','cleaned_text']]
```

OUTPUT:

```
df.head()[['airline_sentiment','text','cleaned_text']]
```

	airline_sentiment	text	cleaned_text
0	neutral	@VirginAmerica What @dhepburn said.	what said.
1	positive	@VirginAmerica plus you've added commercials to the experience... tacky.	plus you've added commercials to the experience... tacky.
2	neutral	@VirginAmerica I didn't today... Must mean I need to take another trip!	i didn't today... must mean i need to take another trip!
3	negative	@VirginAmerica it's really aggressive to blast obnoxious "entertainment" in your guests' faces & they have little recourse	it's really aggressive to blast obnoxious "entertainment" in your guests' faces & they have little recourse
4	negative	@VirginAmerica and it's a really big bad thing about it	and it's a really big bad thing about it

```
# Saving cleaned text and labels to variables
text=df['cleaned_text'].values
labels=df['airline_sentiment'].values
```

```
print(type(text))
print(type(labels))
print(text.shape)
print(labels.shape)
```

OUTPUT:

<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
(14640,)
(14640,)

```
text[50:55]
```

OUTPUT:

array(['is flight 769 on it\'s way? was supposed to take off 30 minutes ago. website still shows "on time" not "in flight". thanks.',
'julie andrews all the way though was very impressive! no to',
'wish you flew out of atlanta... soon?',
'julie andrews. hands down.',
'will flights be leaving dallas for la on february 24th?'],
dtype=object)

```
labels[50:55]
```

OUTPUT:

array(['neutral', 'positive', 'neutral', 'neutral', 'neutral'],dtype=object)

```
# Using label encoder, convert textual labels (positive, negative, neutral) into numners
le=LabelEncoder()

#fit and transform target strings to a number
labels=le.fit_transform(labels)
```

```
le.classes_
```

OUTPUT:

array(['negative', 'neutral', 'positive'], dtype=object)

```
labels
```

OUTPUT:

array(['negative', 'neutral', 'positive'], dtype=object)

```
len(labels)
```

OUTPUT:

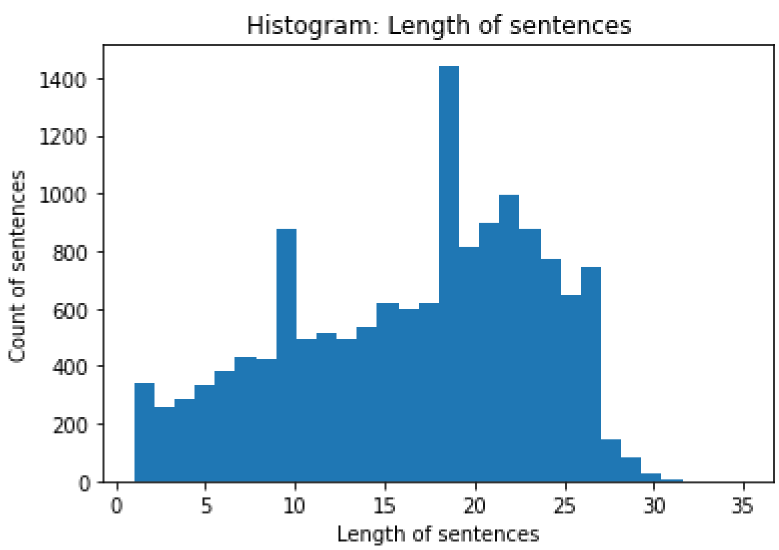
14640

```
#Visualize length of tweets

num=[len(i.split()) for i in text]
plt.hist(num,bins=30)
plt.title('Histogram: Length of sentences')
plt.xlabel('Length of sentences')
plt.ylabel('Count of sentences')
```

OUTPUT:

Text(0, 0.5, 'Count of sentences')



```
#Preparing text input

max_len=28 # This is a hyper parameter which can be tuned
```

```
# Create an empty list to save integer sequence
sent_id=[]

# iterate over each tweet and encode it using bert tokenizer
for i in notebook.tqdm(range(len(text))):
    encoded_sent=tokenizer.encode(text[i],
                                  add_special_tokens=True,
                                  max_length= max_len,
                                  truncation=True,
                                  pad_to_max_length='right'
                                  )

    # save integer sequence to a list
    sent_id.append(encoded_sent)
```

```
print(text[0])
```

OUTPUT:

what said.

```
print(sent_id[0])
```

OUTPUT:

[101, 2054, 2056, 1012, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

```
len(sent_id)
```

OUTPUT:

14640

```
attention_mask=[]

for sent in sent_id:
    attn_mask=[int(token_id>0) for token_id in sent]
    attention_mask.append(attn_mask)
```

```
len(attention_mask)
```

OUTPUT:

14640

```
# Step 6: Training and Valida# Splitting input data
train_inputs,validation_inputs,           train_labels,validation_labels=train_test_split(sent_id,labels,random_state=2018,
test_size=0.1,stratify=labels)
# Splitting masks
train_mask,validation_mask,_,_=
train_test_split(attention_mask,labels,random_state=2018,test_size=0.1,stratify=labels)
```

```
#Step 7: Define Dataloaders
# Converting all inputs and labels into torch tensors which is the required datatype for the BERT model

train_inputs=torch.tensor(train_inputs)
train_labels=torch.tensor(train_labels)
train_mask=torch.tensor(train_mask)
```

```
validation_inputs
```

OUTPUT:

```
tensor([[  101,  2044,  1016,  ...,  22368,  1012,   102],
        [  101,  1045,  2444,  ...,   2068,  1012,   102],
        [  101,  1996,  2034,  ...,   1045,  2081,   102],
        ...,
        [  101,  2003,  2045,  ...,   1012,  1045,   102],
        [  101,  2073,  1005,  ...,      0,     0,     0],
        [  101,  1996,  2711,  ...,      0,     0,     0]])
```

```
# batch size
batch_size=64
# Creating Tensor Dataset for training data
train_data=TensorDataset(train_inputs,train_mask,train_labels)
# Defining a random sampler during training
train_sampler=RandomSampler(train_data)
# Creating iterator using DataLoader. This iterator supports batching, customized data loading order
train_dataloader=DataLoader(train_data,sampler=train_sampler,batch_size=batch_size )
# Creating tensor dataset for validation data
validation_data=TensorDataset(validation_inputs,validation_mask,validation_labels)
# Defining a sequential sampler during validation, bcz there is no need to shuffle the data. We just need to validate
validation_sampler=SequentialSampler(validation_data)
# Create an iterator over validation dataset
validation_dataloader=DataLoader(validation_data,sampler=validation_sampler,batch_size=batch_size)
```

```
# Create an iterator object
iterator=iter(train_dataloader)

# loads batch data
sent_id,mask,target=iterator.__next__()
```

```
sent_id.shape
```

OUTPUT:

```
torch.Size([64, 28])
```

```
sent_id
```

OUTPUT:

```
tensor([[  101,  1045,  2123,  ...,  2340,  2572,   102],
        [  101,  2003,  2045,  ...,  1012,  2053,   102],
        [  101,  1996,  3042,  ..., 16649,  2043,   102],
        ...,
        [  101,  2023,  2003,  ...,  2006,  1037,   102],
        [  101, 18356,  1998,  ...,  2017,  2069,   102],
        [  101,  3462,  8014,  ...,  1055,  2026,   102]])
```

```
outputs=bert(sent_id,attention_mask=mask)
```

```
hidden_states=outputs[0]
CLS_hidden_state=outputs[1]

print("Shape of Hidden States:",hidden_states.shape)
print("Shape of CLS Hidden State:",CLS_hidden_state.shape)
```

OUTPUT:

```
Shape of Hidden States: torch.Size([64, 28, 768])
Shape of CLS Hidden State: torch.Size([64, 768])
```

```
type(hidden_states)
```

OUTPUT:

```
type(hidden_states)
```

```
#Step 8: Fine-Tuning BERT
# turn off the gradient of all parameters

for param in bert.parameters():
   param.requires_grad=False
```

```
class classifier(nn.Module):

    #define the layers and wrappers used by model
    def __init__(self, bert):

      #constructor
      super(classifier, self).__init__()

      #bert model
      self.bert = bert

      # dense layer 1
      self.fc1 = nn.Linear(768,512)

      #dense layer 2 (Output layer)
      self.fc2 = nn.Linear(512,3)

      #dropout layer
      self.dropout = nn.Dropout(0.1)

      #relu activation function
      self.relu =   nn.ReLU()

      #softmax activation function
      self.softmax = nn.LogSoftmax(dim=1)

    #define the forward pass
    def forward(self, sent_id, mask):

      #pass the inputs to the model
      all_hidden_states, cls_hidden_state = self.bert(sent_id, attention_mask=mask, return_dict=False)

      #pass CLS hidden state to dense layer
      x = self.fc1(cls_hidden_state)

      #Apply ReLU activation function
      x = self.relu(x)

      #Apply Dropout
      x = self.dropout(x)

      #pass input to the output
layer
```

```
# create the model
model=classifier(bert)

# push the model to GPU, if available
model=model.to(device)
```

```
# model arcitecture
model
```

```
type(sent_id)
```

OUTPUT:

```
torch.Tensor
```

```
# push the tensors to GPU
sent_id=sent_id.to(device)
mask=mask.to(device)
target=target.to(device)
```

```
# pass inputs to the model
outputs=model(sent_id,mask)
```

```
outputs=outputs.to(device)
```

```
print(outputs)
```

OUTPUT:

```
tensor([[-0.9960, -1.4255, -0.9409],
        [-1.0602, -1.3814, -0.9104],
        [-0.9231, -1.4759, -0.9831],
        [-0.9460, -1.4035, -1.0052],
        [-1.0492, -1.3077, -0.9693],
        [-1.0200, -1.3869, -0.9427],
        [-0.9937, -1.4267, -0.9424],
        [-0.9383, -1.4846, -0.9620],
        [-0.9346, -1.4270, -1.0018],
        [-0.9385, -1.4689, -0.9713],
        [-0.9775, -1.3594, -1.0026],
        [-0.9670, -1.4080, -0.9805],
        [-0.9592, -1.3678, -1.0157],
        [-0.9903, -1.3783, -0.9767],
```

```
[-1.0310, -1.3777, -0.9386],
[-0.9318, -1.4225, -1.0078],
[-1.0117, -1.4004, -0.9418],
[-0.9724, -1.4194, -0.9677],
[-1.0077, -1.3191, -1.0008],
[-1.1239, -1.2910, -0.9163],
[-0.9806, -1.3769, -0.9874],
[-1.0778, -1.4138, -0.8760],
[-0.9843, -1.4388, -0.9440],
[-1.0237, -1.4379, -0.9080],
[-0.9024, -1.4935, -0.9948],
[-1.0459, -1.4401, -0.8874],
[-0.9492, -1.4018, -1.0030],
[-0.8851, -1.4884, -1.0172],
[-0.9620, -1.4861, -0.9375],
[-0.9383, -1.4286, -0.9968],
[-0.9473, -1.4407, -0.9797],
[-1.0908, -1.3356, -0.9136],
[-1.0898, -1.3512, -0.9043],
[-0.9546, -1.4741, -0.9518],
[-1.0250, -1.3957, -0.9325],
[-0.9580, -1.4118, -0.9872],
[-1.0212, -1.2985, -1.0027],
[-1.0807, -1.3390, -0.9200],
[-1.0069, -1.3805, -0.9592],
[-0.9301, -1.4883, -0.9683],
[-1.0115, -1.3251, -0.9927],
[-0.9517, -1.4319, -0.9807],
[-0.9659, -1.3798, -1.0004],
[-0.8912, -1.5049, -1.0002],
[-0.9809, -1.3245, -1.0241],
[-0.9120, -1.4271, -1.0265],
[-0.9053, -1.4309, -1.0314],
[-0.9825, -1.3631, -0.9949],
[-0.9912, -1.4685, -0.9198],
[-0.9859, -1.4011, -0.9660],
[-1.0054, -1.3767, -0.9631],
[-0.9299, -1.3908, -1.0312],
[-0.8797, -1.5794, -0.9702],
[-1.0011, -1.3644, -0.9755],
[-0.9722, -1.3923, -0.9855],
[-0.9424, -1.5133, -0.9413],
[-0.9472, -1.4304, -0.9863],
[-0.9978, -1.4733, -0.9109],
```

```
 [-0.9955, -1.4294, -0.9391],
 [-1.0084, -1.3631, -0.9694],
 [-0.9013, -1.5595, -0.9578],
 [-0.9298, -1.4683, -0.9807],
 [-1.0009, -1.3300, -0.9997],
 [-0.9346, -1.5115, -0.9502]], device='cuda:0',
grad_fn=<LogSoftmaxBackward0>)
```

```
# no. of trainable parameters
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)

print(f'The model has {count_parameters(model):,} trainable parameters')
```

OUTPUT:

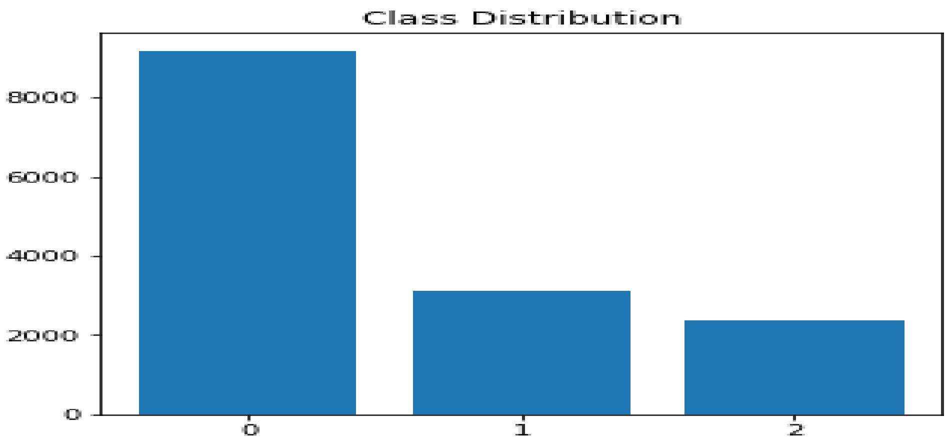
```
The model has 395,267 trainable parameters
```

```
# Adam optmizer
optimizer=torch.optim.Adam(model.parameters(),lr=0.0005)
```

```
# Understnding class distribution
keys=['0','1','2']
# set figure size
plt.figure(figsize=(5,5))
# plot bar chart
plt.bar(keys,class_counts)
# set title
plt.title('Class Distribution')
```

OUTPUT:

```
Text(0.5, 1.0, 'Class Distribution')
```



```
# library for array processing
import numpy as np

# computing the class weights
class_weights=compute_class_weight(class_weight='balanced',classes=np.unique(labels),y=labels)
print("Class Weights:",class_weights)
```

OUTPUT:

```
Class Weights: [0.53170625 1.57470152 2.06517139]
```

```
# Converting a list of class weights into a tensor
weights=torch.tensor(class_weights, dtype=torch.float)

# transferring weights to GPU
weights=weights.to(device)

# define the loss function
cross_entropy=nn.NLLLoss(weight=weights)
```

```
# Computing the loss
print(target)
#print(outputs)
loss=cross_entropy(outputs,target)
print('Loss: ',loss)
```

OUTPUT:

```
tensor([0, 1, 0, 0, 2, 1, 2, 0, 0, 2, 0, 0, 0, 1, 1, 2, 0, 0, 0, 0, 0,
0, 0, 0,
        0, 0, 0, 0, 2, 2, 0, 0, 0, 1, 0, 0, 0, 1, 2, 0, 1, 0, 0, 0, 0,
0, 0, 1,
        0, 2, 1, 0, 0, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0], device='cuda:0')
Loss:  tensor(1.0889, device='cuda:0', grad_fn=<NllLossBackward0>)
```

```
# Function for computing time in hh:mm:ss

def format_time(elapsed):

    elapsed_rounded=int(round(elapsed))

    # format intp hh:mm:ss
    return str(datetime.timedelta(seconds=elapsed_rounded))
```

Training Phase

```
# Defining a training function for the model:
def train():
    print('\n Training')
    # set the model on training phase- Dropout layers are activated
    model.train()
    # recording current time
    t0=time.time()
    # initialize the loss and accuracy to 0
    total_loss,total_accuracy=0,0
    # Create an empty list to save the model prediction
    total_preds=[]
    # for every batch
    for step, batch in enumerate(train_dataloader):
        #Progress update after every 40 batches
        if step % 40==0 and not step==0:
            elapsed=format_time(time.time()-t0)                  # Calculate elapsed time in minutes
            print(' Batch{:>5,} of {:>5,}. Elapsed: {:}.'.format(step,len(train_dataloader),elapsed)) # Print progress
        batch=tuple(t.to(device) for t in batch)           # push the batch to GPU
        # batch is a part of all the records in train_dataloader. It contains 3 pytorch tensors:
        # [0]: input ids
        # [1]: attention masks
        # [2]: labels
        sent_id,mask,labels=batch
  #Pytorch doesn't automatically clear previously calculated gradients, hence before performing a backward pass
        model.zero_grad()
        # Perform a forward pass. This returns the model predictions
        preds=model(sent_id,mask)
        # Compute the loss between actual and predicted values
        loss=cross_entropy(preds,labels)
        #Accumulate training loss over all the batches, so that we can calculate the average loss at the end
        # loss is a tensor containing a single value.
        # .itme() method just returns the Python value from the tensor
        total_loss=total_loss+loss.item()
        # Perform backward pass to calculate the gradients
        loss.backward()
        # During backward pass, information about parameter changes flows backwards, from the output to the hidden
layers to the input
        optimizer.step()
        # Update parameters and take a step using the computed gradient.
        # Here, the optimizer dictates the update rule = how the parameters are modified based on their gradients,
learning rate and so on.
        # The model predictions are stored on GPU, so push it to CPU
        preds=preds.detach().cpu().numpy()
        # Accumulate model predicitons of each batch
        total_preds.append(preds)
  # Compute the training loss of an epoch
    avg_loss=total_loss/len(train_dataloader)
    # The prediction are in the form of (no. of batches, size of batch, no. of classes)
    # So we need to resahpe the predictions in the form of number of samples x number of classes
    total_preds=np.concatenate(total_preds, axis=0)
    return avg_loss,total_preds
```

Evaluation Phase

```
# define a function for evaluating the model

def evaluate():
  print("'n Evaluating....")

  # set the model on validation phase. Here dropout layers are deactivated
  model.eval()

  # record the current time
  t0=time.time()

  # initialize loss and accuracy to 0
  total_loss, total_accuracy=0,0

  # Create an empty list to save model predicitons
  total_preds=[]

  # for each batch

  for step, batch in enumerate(validation_dataloader):
    if step%40==0 and not step ==0:
      elapsed=format_time(time.time()-t0)
      print('   Batch {:>5,}   of   {:>5,}.      Elapsed: {:}.'.format(step, len(validation_dataloader), elapsed))

    batch=tuple(t.to(device) for t in batch)
    sent_id,mask,labels=batch

    #deactivate autograd
    with torch.no_grad():

      preds=model(sent_id,mask)
      loss=cross_entropy(preds,labels)
      total_loss=total_loss+loss.item()
      preds=preds.detach().cpu().numpy()
      total_preds.append(preds)

    avg_loss=total_loss/len(validation_dataloader)

    total_preds=np.concatenate(total_preds,axis=0)

    return avg_loss,total_preds
```

```
#define a function for evaluating the model
def evaluate():
    print("\nEvaluating.....")
    #set the model on training phase - Dropout layers are deactivated
    model.eval()
    #record the current time
    t0 = time.time()
    #initialize the loss and accuracy to 0
    total_loss, total_accuracy = 0, 0
    #Create a empty list to save the model predictions
    total_preds = []
    #for each batch
    for step,batch in enumerate(validation_dataloader):
        # Progress update every 40 batches.
        if step % 40 == 0 and not step == 0:
            # Calculate elapsed time in minutes.
            elapsed = format_time(time.time() - t0)
            # Report progress.
            print('    Batch {:>5,}    of    {:>5,}.        Elapsed: {:}.'.format(step, len(validation_dataloader), elapsed))
        #push the batch to gpu
        batch = tuple(t.to(device) for t in batch)
        #unpack the batch into separate variables
        # `batch` contains three pytorch tensors:
        #      [0]: input ids
        #      [1]: attention masks
        #      [2]: labels
        sent_id, mask, labels = batch
        #deactivates autograd
        with torch.no_grad():
            # Perform a forward pass. This returns the model predictions
            preds = model(sent_id, mask)
            #compute the validation loss between actual and predicted values
            loss = cross_entropy(preds,labels)
            # Accumulate the validation loss over all of the batches so that we can
            # calculate the average loss at the end. `loss` is a Tensor containing a
            # single value; the `.item()` function just returns the Python value
            # from the tensor.
            total_loss = total_loss + loss.item()
            #The model predictions are stored on GPU. So, push it to CPU
            preds=preds.detach().cpu().numpy()
            #Accumulate the model predictions of each batch
            total_preds.append(preds)
    #compute the validation loss of a epoch
    avg_loss = total_loss / len(validation_dataloader)
    #The predictions are in the form of (no. of batches, size of batch, no. of classes).
    #So, reshaping the predictions in form of (number of samples, no. of classes)
    total_preds    = np.concatenate(total_preds, axis=0)

    return avg_loss, total_preds
```

Train the model

```
# Assign the initial loss to infinite
best_valid_loss=float('inf')

# Create an empty list to store training and validation loss of each epoch
train_losses=[]
valid_losses=[]

epochs=5

#for each epoch repeat call the train() method
for epoch in range(epochs):
    print('\n ............epoch {:} / {:} .......'.format(epoch + 1, epochs))

    #train model
    train_loss,_ =train()

    #evaluate model
    valid_loss,_=evaluate()

    # save the best model
    if valid_loss<best_valid_loss:
        best_valid_loss=valid_loss
        torch.save(model.state_dict(),'Saved_weights.pt')

    # Accumulate training and validaion loss
    train_losses.append(train_loss)
    valid_losses.append(valid_loss)

    print(f'\nTraining Loss: {train_loss:.3f}')
    print(f'Validation Loss: {valid_loss:.3f}')

print("")
print("Training complete!")
```

OUTPUT:

```
............epoch 1 / 5 .......

 Training
 Batch   40 of   206. Elapsed: 0:00:04.
 Batch   80 of   206. Elapsed: 0:00:08.
 Batch  120 of   206. Elapsed: 0:00:13.
 Batch  160 of   206. Elapsed: 0:00:17.
 Batch  200 of   206. Elapsed: 0:00:21.

Evaluating.....

Training Loss: 0.716
Validation Loss: 0.746

 ............epoch 2 / 5 .......

 Training
 Batch   40 of   206. Elapsed: 0:00:04.
 Batch   80 of   206. Elapsed: 0:00:09.
 Batch  120 of   206. Elapsed: 0:00:13.
 Batch  160 of   206. Elapsed: 0:00:18.
 Batch  200 of   206. Elapsed: 0:00:22.

Evaluating.....

Training Loss: 0.716
Validation Loss: 0.701

 ............epoch 3 / 5 .......

 Training
 Batch   40 of   206. Elapsed: 0:00:05.
 Batch   80 of   206. Elapsed: 0:00:09.
 Batch  120 of   206. Elapsed: 0:00:14.
 Batch  160 of   206. Elapsed: 0:00:19.
 Batch  200 of   206. Elapsed: 0:00:24.

Evaluating.....
```

```
Training Loss: 0.709
Validation Loss: 0.656

 ............epoch 4 / 5 .......

 Training
 Batch   40 of   206. Elapsed: 0:00:05.
 Batch   80 of   206. Elapsed: 0:00:09.
 Batch  120 of   206. Elapsed: 0:00:14.
 Batch  160 of   206. Elapsed: 0:00:18.
 Batch  200 of   206. Elapsed: 0:00:23.

Evaluating.....

Training Loss: 0.705
Validation Loss: 0.645

 ............epoch 5 / 5 .......

 Training
 Batch   40 of   206. Elapsed: 0:00:04.
 Batch   80 of   206. Elapsed: 0:00:09.
 Batch  120 of   206. Elapsed: 0:00:13.
 Batch  160 of   206. Elapsed: 0:00:18.
 Batch  200 of   206. Elapsed: 0:00:22.

Evaluating.....

Training Loss: 0.705
Validation Loss: 0.640

Training complete!
```

```
# Converting the log(probabilities) into class & then choosing index of maximum value as class
y_pred=np.argmax(preds,axis=1)

# actual labels
y_true=validation_labels
```

Evaluate the model

```
# load weights of best model
path='Saved_weights.pt'
model.load_state_dict(torch.load(path))
```

OUTPUT:

<All keys matched successfully>

```
# get the model prediction on the validation data
valid_loss, preds=evaluate()
# this returns 2 elements- Validation loss and prediction
print(valid_loss)
```

OUTPUT

Evaluating.....

0.6396074994750645

```
print(classification_report(y_true,y_pred))
```

OUTPUT:

	precision	recall	f1-score	support
0	0.93	0.66	0.78	918
1	0.49	0.70	0.58	310
2	0.56	0.89	0.69	236
accuracy			0.71	1464
macro avg	0.66	0.75	0.68	1464
weighted avg	0.78	0.71	0.72	146

Part III PROGRAM - roBERTa

```
import torch
import pandas as pd
import numpy as np
import re
from sklearn.model_selection import train_test_split
pd.set_option('display.max_colwidth',200)
from sklearn.preprocessing import LabelEncoder

import matplotlib.pyplot as plt

#library for progress bar
from tqdm import notebook
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler

# importing nn module
import torch.nn as nn

#library for computing class weights
from sklearn.utils.class_weight import compute_class_weight

from sklearn.metrics import classification_report
import time
import datetime
```

```
# Checking if GPU is available.
if torch.cuda.is_available():
    device=torch.device('cuda')
```

```
print(device)
torch.cuda.get_device_name(0)
# Current GPU is Tesla T4
```

```
:# Step 2: Installing Hugging Face's Transformers Library
# Hugging face is one of the most popular NLP library and provides a wide range of transformer-based models such as BERT, GPT-2, Roberta, and so on.

!pip install transformers
```

```
# Step 3: Installing roBerta Model
rom transformers import RobertaModel, RobertaTokenizer

# Download RoBERTa base model
roberta = RobertaModel.from_pretrained('roberta-base')
tokenizer = RobertaTokenizer.from_pretrained('roberta-base')
```

```
# Print the model architecture
print(roberta)
```

OUTPUT

```
RobertaModel(
  (embeddings): RobertaEmbeddings(
    (word_embeddings): Embedding(50265, 768, padding_idx=1)
    (position_embeddings): Embedding(514, 768, padding_idx=1)
    (token_type_embeddings): Embedding(1, 768)
    (LayerNorm): LayerNorm((768,), eps=1e-05,
elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (encoder): RobertaEncoder(
    (layer): ModuleList(
      (0-11): 12 x RobertaLayer(
        (attention): RobertaAttention(
          (self): RobertaSelfAttention(
            (query): Linear(in_features=768, out_features=768,
bias=True)
            (key): Linear(in_features=768, out_features=768,
bias=True)
            (value): Linear(in_features=768, out_features=768,
bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): RobertaSelfOutput(
            (dense): Linear(in_features=768, out_features=768,
bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-05,
elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
```

```
          )
        )
        (intermediate): RobertaIntermediate(
          (dense): Linear(in_features=768, out_features=3072,
bias=True)
          (intermediate_act_fn): GELUActivation()
        )
        (output): RobertaOutput(
          (dense): Linear(in_features=3072, out_features=768,
bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-05,
elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
  )
  (pooler): RobertaPooler(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (activation): Tanh()
  )
)
```

```
#Step 4: Importing BERT tokenizer

from transformers import RobertaTokenizerFast

tokenizer = RobertaTokenizerFast.from_pretrained('roberta-base', do_lower_case=True)
```

```
# Define your text
text = 'Jim Henson was a puppeteer'

# Define your text
text = 'Jim Henson was a puppeteer'

# Import and initialize RoBERTa tokenizer
tokenizer = RobertaTokenizerFast.from_pretrained('roberta-base',
do_lower_case=True)

# Tokenize and encode the text
sentence_id = tokenizer.encode(text,
                          # Add special tokens
                          add_special_tokens=True,
                          # Specify maximum length for any input sequences
                          max_length=10,
```

OUTPUT:

Integer Sequence: [0, 24021, 289, 13919, 21, 10, 32986, 9306, 254, 2]

```
converting integers back to text
print("Tokenizer Text: ",tokenizer.convert_ids_to_tokens(sentence_id))
```

OUTPUT:

Tokenizer Text: ['<s>', 'Jim', 'ĠH', 'enson', 'Ġwas', 'Ġa', 'Ġpupp', 'ete', 'er', '</s>']

```
decoded=tokenizer.decode(sentence_id)
print('Decoded String:{}'.format(decoded))
```

OUTPUT:

Decoded String:<s>Jim Henson was a puppeteer</s>

```
att_mask=[int(tok>0) for tok in sentence_id]
print(att_mask)
```

OUTPUT:

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

```
Understanding Input and Output of BERT Tokenizer
# convert lists to tensors
# torch.tensor creates a tensor of given data
sent_id=torch.tensor(sentence_id)
attn_mask=torch.tensor(att_mask)
print('Shape of sentence_id before reshaping is: {}'.format(sent_id.shape))
print('Shape of sentence_id before reshaping is: {}'.format(attn_mask.shape))
print('\n')
# reshaping tensor in form of batch,text length
sent_id=sent_id.unsqueeze(0)
attn_mask=attn_mask.unsqueeze(0)
print('Shape of sentence_id after reshaping is: {}'.format(sent_id.shape))
print('Shape of sentence_id after reshaping is: {}'.format(attn_mask.shape))
print('\n')
# reshaped tensor
print(sent_id)
```

OUTPUT:

```
Shape of sentence_id before reshaping is: torch.Size([10])
Shape of sentence_id before reshaping is: torch.Size([10])


Shape of sentence_id after reshaping is: torch.Size([1, 10])
Shape of sentence_id after reshaping is: torch.Size([1, 10])


tensor([[    0, 24021,   289, 13919,    21,    10, 32986,  9306,   254,
2]])
```

```
# passing integer sequence and attention mask tensor to BERT model
out[illegible]roberta(sent_id,attention_mask=attn_mask)
```

```
# Unpacking the output of BERT model

# all_hidden_states is a collection of all the output vectors/ hidden states (of encoder) at each timestamps or position of the BERT model
all_hidden_states=outputs[0]

print(all_hidden_states.shape)
print(all_hidden_states)
```

OUTPUT:

```
torch.Size([1, 10, 768])
tensor([[[-0.0677,  0.0641, -0.0276,  ..., -0.1516, -0.0074,
-0.0088],
         [-0.0347,  0.0307,  0.0616,  ..., -0.6778,  0.2086,
-0.1319],
         [-0.0743,  0.0951, -0.0621,  ..., -0.0301,  0.1281,
0.0751],
         ...,
         [-0.0966,  0.0557, -0.0049,  ..., -0.1686, -0.0279, -0.0405],
         [-0.0678,  0.0622, -0.0365,  ..., -0.1748, -0.0106, -0.0273],
         [-0.0657,  0.0637, -0.0431,  ..., -0.1725, -0.0118,
-0.0270]]],
```

```
# this output contains output vector against the CLS token only (at the first position of BERT model)
# this output vector encodes the entire input sequence

cls_hidden_state=outputs[1]
print(cls_hidden_state.shape)
print(cls_hidden_state)
```

```
grad_fn=<NativeLayerNormBackward0>)
```

OUTPUT:

```
torch.Size([1, 768])
tensor([[ 1.8997e-01,  9.1629e-02, -3.9820e-01,  3.3366e-01,  1.6496e-01,
          2.3452e-01,  1.4336e-01, -5.4239e-02, -8.5553e-02,  2.3895e-02,
         -4.9157e-01,  7.6284e-02, -5.0789e-01,  3.5709e-01, -1.3758e-01,
         -1.1414e-01,  8.7795e-03,  4.3455e-01,  2.6406e-01,  5.9045e-03,
          1.1315e-01, -4.4567e-02, -2.5890e-01,  1.7075e-01,  1.7518e-01,
         -2.4743e-02,  3.7533e-02,  1.8175e-03,  3.3750e-01,  5.4122e-02,
         -9.0557e-02,  5.3360e-02, -2.4335e-01, -1.0459e-01,  4.3863e-04,
          3.0356e-01, -2.2620e-01, -9.6111e-02, -2.1011e-01, -1.3769e-01,
          1.9942e-01, -3.3582e-02,  1.1632e-01,  5.2396e-01,  1.9179e-01,
         -1.7475e-01, -4.9645e-02, -1.6517e-01,  1.1105e-01,  4.2198e-01,
         -3.7382e-01,  4.8912e-01,  1.3690e-02,  5.6702e-01, -1.9044e-02,
         -2.1632e-02, -1.7647e-01, -3.0004e-01, -2.1024e-01,  9.8124e-02,
         -1.3139e-01,  2.6441e-01, -2.7652e-01,  1.3847e-01,  1.1151e-01,
          5.8956e-02, -2.5439e-01,  3.5740e-01,  3.2645e-01,  8.6073e-02,
         -2.8824e-01,  3.2004e-01,  5.5328e-02,  1.5051e-01, -1.7616e-01,
          2.5925e-01, -1.1205e-02,  2.5279e-01, -1.0162e-01, -3.3001e-01,
         -1.2163e-01,  1.3442e-01,  1.0385e-01,  1.9183e-02, -4.7227e-01,
          1.0795e-02, -6.2856e-02,  5.0762e-02, -1.2960e-01, -1.9277e-01,
         -3.5209e-01,  1.2755e-01,  1.5776e-01, -3.7265e-02,  2.6807e-02,
         -2.6721e-01,  1.7658e-01,  2.3152e-01,  4.7825e-01, -1.6338e-01,
         -4.3599e-02, -3.4267e-02,  4.4145e-01, -3.1475e-02,  3.2136e-01,
          3.4149e-02, -2.8820e-02,  2.7718e-01, -1.9999e-01, -3.6943e-01,
         -3.5369e-02, -1.7624e-01,  8.7009e-02,  2.5920e-01, -2.3576e-01,
         -3.4668e-02,  1.1171e-01, -2.3946e-02, -4.5769e-01, -3.3842e-01,
         -1.9056e-01,  1.8479e-01, -2.6213e-01,  1.6036e-01, -4.2248e-01,
          2.1129e-02,  2.0425e-01, -1.3407e-01,  2.4805e-01, -3.2762e-01,
         -2.0293e-01,  6.9899e-02,  3.5062e-01, -1.1153e-01, -3.0906e-01,
         -3.5833e-01,  6.5807e-02, -3.3524e-01,  7.8695e-02,  4.6506e-01,
          5.5567e-02,  1.2390e-01, -1.7104e-01, -1.2483e-01,  3.3041e-02,
         -2.8627e-01,  8.9307e-02,  1.0262e-01,  3.0443e-02, -1.2543e-01,
         -7.4727e-02, -4.0890e-01,  3.9047e-02,  3.5377e-01,  8.2841e-02,
          1.6276e-01, -3.7081e-01, -9.3195e-03, -3.2193e-01, -7.4795e-02,
```

5.1032e-02, -2.1958e-01, 1.5960e-01, -8.4593e-02, 6.5329e-02,
-2.1460e-01, 9.8126e-02, -4.0380e-01, 4.1332e-01, 6.7217e-02,
-2.8337e-01, 2.1758e-02, 1.8059e-01, 3.2666e-01, -8.9008e-03,
-1.0422e-01, 1.9954e-01, -1.1970e-01, 3.1879e-01, -4.3607e-01,
4.9854e-01, 1.2519e-01, -1.5068e-01, -8.0202e-02, -1.4078e-01,
-1.2734e-01, 2.6987e-01, -4.8633e-02, 1.4964e-01, -7.7898e-03,
2.9020e-02, 3.3984e-02, 1.9001e-01, -4.2302e-02, -3.7674e-01,
2.7347e-01, -3.3448e-02, -4.4320e-02, -2.2579e-01, -2.4176e-01,
-2.3236e-01, -1.1597e-01, -2.2631e-01, 1.2753e-02, -4.8693e-01,
-2.2800e-01, 1.9421e-01, -3.3144e-02, -1.5081e-01, 3.8326e-01,
1.9148e-02, 2.5320e-01, -4.7621e-01, -6.3601e-02, -1.7255e-01,
-1.8577e-01, 6.5021e-02, 1.0511e-01, -1.1932e-01, -7.1593e-02,
-9.2873e-02, 1.4269e-02, 3.6082e-01, -3.8744e-02, 3.6412e-02,
2.6790e-01, 3.1500e-01, 8.7378e-02, -7.9673e-02, 1.1789e-01,
-3.3768e-01, 2.5109e-01, 1.0887e-03, 1.4703e-01, 9.3624e-02,
3.6377e-01, -3.2038e-02, 9.9922e-02, 1.9357e-01, -1.1006e-01,
2.8055e-01, 9.6841e-02, -9.3571e-03, -1.9134e-01, 1.6599e-01,
-9.7727e-02, 6.0381e-02, -3.2750e-01, -9.4595e-02, -1.2063e-01,
-9.4217e-02, 2.5026e-01, 8.9594e-02, -2.2733e-01, -8.4628e-04,
4.7572e-01, 1.0510e-01, 2.3752e-01, 2.1211e-01, -4.9846e-02,
-2.2938e-01, -9.8428e-02, -2.4041e-01, 2.9925e-01, -2.5548e-01,
9.6122e-02, -1.3744e-02, -5.1914e-02, 1.6303e-01, -1.1896e-01,
-3.8868e-01, -9.0995e-02, -2.5836e-02, -4.4282e-01, -1.4443e-01,
-3.5419e-01, -6.9252e-02, 8.3126e-02, 5.4915e-01, 3.9838e-01,
1.2727e-01, -4.3324e-01, 6.6434e-02, -2.4480e-02, 9.9093e-02,
4.1645e-02, -3.1794e-01, 5.9664e-02, -5.5013e-02, 1.6059e-01,
-3.0060e-01, 1.0867e-01, -2.0745e-01, -1.5732e-01, -2.3847e-01,
-4.7874e-02, 2.7384e-01, -1.4496e-01, 3.4230e-01, 2.9477e-02,
7.4363e-02, 1.0406e-01, 1.3667e-01, 2.2802e-01, -9.8962e-02,
1.6611e-01, 5.4884e-01, 5.4427e-02, 1.1582e-01, 1.2375e-02,
-4.0954e-02, -1.1322e-01, 7.1990e-02, 2.1165e-01, -3.4982e-01,
-1.2596e-01, -1.8859e-01, 5.0409e-02, -1.9397e-01, -1.2991e-01,
1.8206e-01, -6.9503e-04, 6.7991e-02, 2.0638e-01, -1.9522e-01,
-2.2749e-01, -1.7358e-01, 1.5319e-02, 8.5513e-02, -3.2841e-01,
-9.1695e-02, 4.3937e-02, -8.1126e-02, 1.2159e-01, -2.4001e-01,
6.8346e-02, 1.8098e-01, 1.6805e-01, -2.0134e-01, 2.2459e-02,
-3.1700e-02, 2.0395e-02, 2.2864e-02, 3.3604e-02, 1.6855e-01,
2.5853e-02, 3.9846e-01, 4.2257e-01, -2.0332e-01, -3.4885e-01,
1.9194e-02, 1.5062e-01, -3.1147e-01, 1.3942e-01, 3.0127e-01,
-2.1043e-01, -1.8772e-02, -1.1990e-01, -3.1167e-01, -2.0649e-01,
3.3933e-01, -2.3174e-01, 5.6309e-02, 1.5435e-01, -4.4586e-02,
2.9518e-02, -1.8808e-01, -9.8883e-02, 1.7236e-02, -2.9747e-01,
4.7238e-01, 2.2315e-02, 3.1837e-01, 6.1942e-02, 4.1071e-01,
-2.7678e-02, -1.3466e-01, -2.0653e-01, -6.9751e-02, 9.3838e-02,

3.4654e-01, -4.0870e-01, 1.6786e-01, -3.9483e-01, -7.6481e-02,
-4.1322e-02, -7.4650e-02, 2.7280e-01, -1.1193e-01, -1.1713e-01,
1.2780e-01, 5.3973e-02, 6.2404e-02, 9.7770e-02, 1.5040e-01,
-2.0606e-01, 4.7294e-01, -8.1126e-02, 8.4722e-02, -3.5042e-01,
2.0288e-01, -9.2740e-02, -3.5827e-02, 1.3635e-01, -1.7683e-01,
3.0951e-01, -1.7663e-01, 1.4552e-01, 1.9281e-01, 2.6019e-02,
-1.8753e-01, 9.3978e-02, -4.6764e-02, 2.3134e-01, -1.8911e-01,
1.4133e-01, 1.6251e-01, -5.8301e-02, -4.1067e-02, 1.0730e-01,
2.1981e-01, 6.6432e-02, -2.2411e-01, -2.6068e-01, -3.2171e-01,
2.7420e-01, 1.1662e-01, -8.4478e-02, 8.5552e-02, -1.3560e-01,
1.4615e-01, -1.0755e-01, 1.5422e-01, -1.8084e-01, -1.5503e-01,
-1.3854e-01, 2.0314e-01, -3.5579e-02, -6.0765e-02, 2.5821e-03,
4.4933e-02, -6.8552e-02, -2.5014e-01, -1.1783e-01, 1.7108e-01,
-2.5271e-01, -1.4025e-01, -5.6493e-02, 3.4818e-01, -1.0089e-01,
-1.8502e-01, -2.5331e-01, 3.9773e-01, 4.1064e-02, -5.3153e-02,
-1.0753e-02, -1.8528e-02, -9.8683e-02, -3.9880e-01, 4.7883e-01,
-9.6845e-02, -2.4630e-01, -1.3856e-01, 1.9648e-01, -1.0844e-01,
3.0865e-04, 1.7011e-02, -7.2891e-02, -1.7712e-01, 3.8018e-02,
-2.7656e-02, -1.5044e-01, -3.9446e-02, 3.1890e-01, -2.4729e-01,
2.0273e-01, 5.3312e-02, 1.2290e-01, -1.9205e-01, -1.7685e-01,
5.2505e-02, -6.0671e-02, -1.3913e-02, -2.0945e-01, -9.8684e-03,
8.6467e-02, -4.4032e-01, 1.3349e-01, -7.3085e-02, 5.2446e-02,
1.8022e-01, -3.3222e-01, 1.0767e-01, 1.5785e-01, -9.1950e-02,
5.3685e-01, 9.6555e-02, 1.1743e-01, 3.7780e-01, -1.5715e-01,
-2.5121e-01, 3.8022e-01, -3.5128e-02, 3.0788e-01, -3.6025e-01,
-1.9865e-01, 3.4383e-01, 7.6263e-02, -1.6911e-01, 1.8290e-01,
-5.2965e-01, -3.8107e-01, 6.1385e-02, -5.1499e-02, -3.9958e-01,
-1.2701e-01, -1.9914e-02, 3.3166e-01, 2.9204e-01, 6.5472e-02,
1.5002e-01, -3.5668e-01, 1.2888e-02, 3.3804e-01, -7.1830e-02,
-5.6954e-02, -2.3058e-01, 2.1036e-01, 8.5141e-02, 3.4098e-02,
2.2036e-01, 1.4443e-01, 7.7125e-02, -1.7082e-01, -2.8648e-02,
1.6978e-01, -2.3125e-01, -1.1658e-02, 1.1215e-01, -2.9998e-01,
-1.0153e-01, -1.5806e-01, -2.9623e-01, -1.7352e-01, 7.4652e-03,
-7.8402e-02, -1.6425e-02, 7.1718e-02, -1.0421e-01, 2.8365e-01,
-8.1096e-02, 1.8127e-01, 2.9512e-01, 1.7342e-01, 3.4592e-01,
4.2097e-02, -1.2317e-01, -8.4213e-02, 1.5245e-01, -4.3074e-01,
1.5893e-01, -1.2912e-01, 4.5504e-01, -3.3937e-01, -1.0143e-02,
-4.2167e-02, 4.0437e-01, 1.2330e-01, 2.9812e-01, 7.5590e-02,
-6.4105e-02, -2.0080e-01, -5.8558e-02, -5.2861e-02, -1.7070e-01,
-3.1215e-01, -2.6469e-01, -9.5930e-03, -2.6621e-01, 1.7055e-01,
3.2471e-01, 2.9457e-01, 9.2084e-02, -3.5188e-01, 1.9144e-02,
-2.8379e-01, -1.6125e-01, 1.5014e-01, -8.3844e-02, -1.0368e-01,
-2.1699e-01, -1.6836e-01, 1.7384e-01, 7.2089e-02, 3.0076e-01,
-1.8368e-01, -3.4251e-01, -2.8702e-01, -7.4484e-02, 9.5774e-02,

```
-8.5355e-03, -5.4892e-02, -1.1980e-01,  2.7121e-02,  3.6681e-01,
-1.2239e-01,  2.8694e-03, -2.8034e-01, -1.8706e-01, -3.6829e-01,
-1.7769e-01, -1.9263e-01, -3.1230e-01,  6.0145e-03,  1.7524e-01,
-1.7385e-01, -1.1213e-01, -7.7062e-02, -2.7075e-01, -9.6823e-02,
-1.5774e-01,  1.1521e-01,  4.2841e-01,  2.4049e-01, -2.4422e-02,
 1.7693e-01,  1.0302e-01, -3.9279e-01, -2.7301e-02,  9.1910e-02,
 1.7560e-01,  3.1260e-01, -2.3886e-02,  1.7863e-01, -1.7165e-01,
 2.3338e-01,  6.6140e-01, -1.8433e-01, -1.2154e-01,  1.1330e-01,
 6.8499e-03,  2.4543e-01, -2.1893e-01,  5.5481e-01, -3.3689e-01,
 2.0154e-01, -3.0396e-01,  7.8017e-03,  2.9096e-01,  1.5601e-01,
-4.9816e-02,  4.3000e-02, -2.6273e-01, -1.2473e-01, -2.2216e-01,
 2.9929e-02,  1.0615e-01, -1.3580e-01,  7.1381e-02,  7.0862e-02,
-4.0845e-02, -5.4554e-01,  1.0194e-01,  1.8555e-01, -6.7282e-02,
 2.0953e-01,  3.9374e-02, -3.1091e-01,  4.0083e-01, -2.9818e-01,
 2.2702e-01, -4.4082e-02,  1.0840e-01,  4.9286e-02,  1.2481e-02,
 2.0824e-01, -3.4434e-02, -4.1012e-01, -2.3324e-01, -2.5164e-02,
 4.7014e-01, -1.8155e-01, -9.9913e-03, -1.2398e-01, -2.4707e-01,
-2.1755e-01,  3.0213e-01, -1.8790e-02,  8.1139e-02,  1.1371e-02,
-3.6812e-01, -1.0906e-01, -1.0203e-01,  2.4310e-01,  1.2677e-01,
-1.7457e-01,  7.8277e-02,  4.4643e-02, -4.4524e-02, -1.2948e-02,
-1.0342e-02,  5.0975e-03, -4.1783e-01, -1.5808e-01, -1.8620e-01,
 1.3305e-01,  1.4745e-01,  7.4080e-02,  5.6626e-02,  2.2239e-01,
-8.4726e-02, -2.0123e-01, -4.7728e-02, -2.6117e-01,  6.3750e-02,
 3.0807e-01, -1.5645e-01, -9.1572e-02, -9.2749e-02, -2.1286e-01,
 2.5461e-02, -4.6822e-02,  1.1741e-02,  1.1012e-01, -1.3210e-01,
 3.7935e-01, -4.6380e-01,  4.0456e-01, -8.2525e-02,  6.0233e-02,
 1.2237e-01, -5.8600e-02,  3.3111e-01, -3.4757e-01, -1.0543e-01,
 2.0734e-01,  2.9905e-01,  1.6301e-01, -6.7869e-02, -9.6951e-02,
 1.3971e-01,  1.2596e-01, -7.6262e-03, -4.5770e-01,  1.3966e-01,
 5.6120e-02,  1.1937e-01,  5.7955e-02,  1.2572e-01, -2.4188e-01,
 2.1657e-01, -4.9960e-01,  3.8204e-01, -2.9365e-01, -1.3000e-01,
 1.0173e-01,  3.3060e-01, -2.2820e-01, -1.6854e-01, -2.1891e-01,
-3.6740e-01,  1.2428e-01, -6.5428e-02,  7.5426e-02,  1.7724e-01,
 1.2524e-01, -1.8985e-01, -2.8223e-01]], grad_fn=<TanhBackward0>)
```

```
#Step 5: Data Preparation

df=pd.read_csv('Tweets.csv')
df.head()
```

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline	airline_sentiment_gold	name	negativereason_gold	retweet_count	text	tweet_coord	tweet_created	tweet_location	user_timezone
0	570306133677760513	neutral	1.0000	NaN	NaN	Virgin America	NaN	cairdin	NaN	0	@VirginAmerica What @dhepburn said.	NaN	2015-02-24 11:35:52 -0800	NaN	Eastern Time (US & Canada)
1	570301130888122368	positive	0.3486	NaN	0.0000	Virgin America	NaN	jnardino	NaN	0	@VirginAmerica plus you've added commercials to the experience... tacky.	NaN	2015-02-24 11:15:59 -0800	NaN	Pacific Time (US & Canada)
2	570301083672813571	neutral	0.6837	NaN	NaN	Virgin America	NaN	yvonnalynn	NaN	0	@VirginAmerica I didn't today... Must mean I need to take another trip!	NaN	2015-02-24 11:15:48 -0800	Lets Play	Central Time (US & Canada)
3	570301031407624196	negative	1.0000	Bad Flight	0.7033	Virgin America	NaN	jnardino	NaN	0	@VirginAmerica it's really aggressive to blast obnoxious "entertainment" in your guests' faces & they have little recourse	NaN	2015-02-24 11:15:36 -0800	NaN	Pacific Time (US & Canada)
4	570300817074462722	negative	1.0000	Can't Tell	1.0000	Virgin America	NaN	jnardino	NaN	0	@VirginAmerica and it's a really big bad thing about it	NaN	2015-02-24 11:14:45 -0800	NaN	Pacific Time (US & Canada)

```
df.shape
```

OUTPUT:

(14640, 15)

Distribution of Tweets (label)

```
print(df['airline_sentiment'].value_counts())
print(df['airline_sentiment'].value_counts(normalize=True))
```

OUTPUT:

negative 9178
neutral 3099
positive 2363
Name: airline_sentiment, dtype: int64
negative 0.626913
neutral 0.211680
positive 0.161407
Name: airline_sentiment, dtype: float64

```
# Sabing value counts to a list
class_counts=df['airline_sentiment'].value_counts().to_list()
```

- Removing twitter usernames
- Removing links (starting with https)

```
def preprocess(text):
    # converting text tolower case
    text=text.lower()
    # remove user mentions
    text=re.sub(r'@[A-Za-z0-9]+','',text)
    # remove hashtags if needed keep for now
    #text=re.sub(r'#[A-Za-z0-9]+','',text)
    # remove links
    text=re.sub(r'http\S+','',text)
    # Split tokens so that extra spaces which were added due to above substitution are removed
    tokens=text.split()

    # join tokens by space
    return ' '.join(tokens)
```

```
# using apply function to apply this preprocess function on each row of the text column
df['cleaned_text']=df['text'].apply(preprocess)
```

```
df.head()[['airline_sentiment','text','cleaned_text']]
```

```
print(type(text))
print(type(labels))
print(text.shape)
print(labels.shape)
```

OUTPUT

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
(14640,)
(14640,)
```

Preparing input and output data

- **Preparing target input**

```
df.head()[['airline_sentiment','text','cleaned_text']]
```

```
le.classes_
```

OUTPUT:

array(['negative', 'neutral', 'positive'], dtype=object)

```
labels
```

OUTPUT:

array([1, 2, 1, ..., 1, 0, 1])

```
len(labels)
```

OUTPUT:

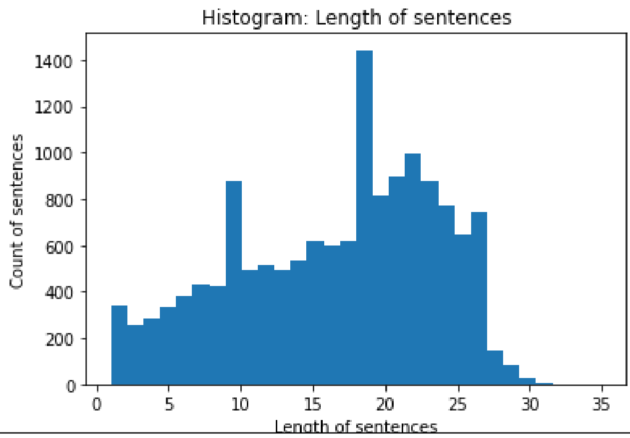
14640

```
#Visualize length of tweets

num=[len(i.split()) for i in text]
plt.hist(num,bins=30)
plt.title('Histogram: Length of sentences')
plt.xlabel('Length of sentences')
plt.ylabel('Count of sentences')
```

OUTPUT:

Text(0, 0.5, 'Count of sentences')




```
#Preparing text input

max_len=28 # This is a hyper parameter which can be tuned
```

```
# Create an empty list to save integer sequence
sent_id=[]

# iterate over each tweet and encode it using bert tokenizer
for i in notebook.tqdm(range(len(text))):
    encoded_sent=tokenizer.encode(text[i],
                                  add_special_tokens=True,
                                  max_length= max_len,
                                  truncation=True,
                                  pad_to_max_length='right'
                                  )

    # save integer sequence to a list
    sent_id.append(encoded_sent)
```

```
print(text[0])
```

OUTPUT:

```
what said.
```

```
print(sent_id[0])
```

OUTPUT:

[0, 12196, 26, 4, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

```
len(sent_id)
```

OUTPUT:

14640

```
attention_mask=[]

for sent in sent_id:
    attn_mask=[int(token_id>0) for token_id in sent]
    attention_mask.append(attn_mask)
```

```
len(attention_mask)
```

OUTPUT:

0

```
# Step 6: Training and Valida# Splitting input data
train_inputs,validation_inputs,            train_labels,validation_labels=train_test_split(sent_id,labels,random_state=2018,
test_size=0.1,stratify=labels)
# Splitting masks
train_mask,validation_mask,_,_=
train_test_split(attention_mask,labels,random_state=2018,test_size=0.1,stratify=labels)
```

```
#Step 7: Define Dataloaders
# Converting all inputs and labels into torch tensors which is the required datatype for the BERT model

train_inputs=torch.tensor(train_inputs)
train_labels=torch.tensor(train_labels)
train_mask=torch.tensor(train_mask)
```

```
validation_inputs
```

OUTPUT:

```
tensor([[ 0, 10669, 132, ..., 4, 14, 2], [ 0, 118, 697, ..., 24, 74,
2], [ 0, 627, 78, ..., 156, 5, 2], ..., [ 0, 354, 89, ..., 939, 240,
2], [ 0, 8569, 18, ..., 1, 1, 1], [ 0, 627, 621, ..., 1, 1, 1]])
```

```
# batch size
batch_size=64
# Creating Tensor Dataset for training data
train_data=TensorDataset(train_inputs,train_mask,train_labels)
# Defining a random sampler during training
train_sampler=RandomSampler(train_data)
# Creating iterator using DataLoader. This iterator supports batching, customized data loading order
train_dataloader=DataLoader(train_data,sampler=train_sampler,batch_size=batch_size )
# Creating tensor dataset for validation data
validation_data=TensorDataset(validation_inputs,validation_mask,validation_labels)
# Defining a sequential sampler during validation, bcz there is no need to shuffle the data. We just need to validate
validation_sampler=SequentialSampler(validation_data)
# Create an iterator over validation dataset
validation_dataloader=DataLoader(validation_data,sampler=validation_sampler,batch_size=batch_size)
```

```
# Create an iterator object
iterator=iter(train_dataloader)

# loads batch data
sent_id,mask,target=iterator.__next__()
```

```
sent_id.shape
```

OUTPUT:

```
torch.Size([64, 28])
```

```
sent_id
```

OUTPUT:

```
tensor([[ 0, 12459, 15, ..., 116, 2, 1], [ 0, 2362, 6, ..., 2, 1, 1],
[ 0, 4182, 5361, ..., 1, 1, 1], ..., [ 0, 8987, 77, ..., 1, 1, 1],
[ 0, 2456, 1902, ..., 2, 1, 1], [ 0, 12, 1948, ..., 360, 4, 2]])
```

```
outputs=bert(sent_id,attention_mask=mask)
```

```
hidden_states=outputs[0]
CLS_hidden_state=outputs[1]

print("Shape of Hidden States:",hidden_states.shape)
print("Shape of CLS Hidden State:",CLS_hidden_state.shape)
```

OUTPUT

```
Shape of Hidden States: torch.Size([64, 28, 768])
Shape of CLS Hidden State: torch.Size([64, 768])
```

```
#Step 8: Fine-Tuning BERT
# turn off the gradient of all parameters

for param in roberta.parameters():
    param.requires_grad=False
```

```
class Classifier(nn.Module):
        def __init__(self, roberta):
                super(Classifier, self).__init__()
                self.roberta = roberta
                self.fc1 = nn.Linear(768, 512)
                self.fc2 = nn.Linear(512, 3)
                self.dropout = nn.Dropout(0.1)
                self.relu = nn.ReLU()
                self.softmax = nn.LogSoftmax(dim=1)

        def forward(self, input_ids, attention_mask):
                outputs = self.roberta(input_ids, attention_mask=attention_mask)
                cls_hidden_state = outputs.last_hidden_state[:, 0, :]    # Get the CLS token hidden state
                x = self.fc1(cls_hidden_state)
                x = self.relu(x)
                x = self.dropout(x)
                x = self.fc2(x)
                x = self.softmax(x)
                return x
```

```
# create the model
model=classifier(roberta)

# push the model to GPU, if available
model=model.to(device)
```

```
# model arcitecture
model
```

```
type(sent_id)
```

OUTPUT:

```
torch.Tensor
```

```
# push the tensors to GPU
sent_id=sent_id.to(device)
mask=mask.to(device)
target=target.to(device)
```

```
# pass inputs to the model
outputs=model(sent_id,mask)
```

```
outputs=outputs.to(device)
```

```
print(outputs)
```

OUTPUT:

```
tensor([[-0.9960, -1.4255, -0.9409],
        [-1.0602, -1.3814, -0.9104],
        [-0.9231, -1.4759, -0.9831],
        [-0.9460, -1.4035, -1.0052],
        [-1.0492, -1.3077, -0.9693],
        [-1.0200, -1.3869, -0.9427],
        [-0.9937, -1.4267, -0.9424],
        [-0.9383, -1.4846, -0.9620],
        [-0.9346, -1.4270, -1.0018],
        [-0.9385, -1.4689, -0.9713],
        [-0.9775, -1.3594, -1.0026],
        [-0.9670, -1.4080, -0.9805],
        [-0.9592, -1.3678, -1.0157],
        [-0.9903, -1.3783, -0.9767],
        [-1.0310, -1.3777, -0.9386],
        [-0.9318, -1.4225, -1.0078],
        [-1.0117, -1.4004, -0.9418],
        [-0.9724, -1.4194, -0.9677],
        [-1.0077, -1.3191, -1.0008],
        [-1.1239, -1.2910, -0.9163],
        [-0.9806, -1.3769, -0.9874],
```

```
 [-1.0778, -1.4138, -0.8760],
 [-0.9843, -1.4388, -0.9440],
 [-1.0237, -1.4379, -0.9080],
 [-0.9024, -1.4935, -0.9948],
 [-1.0459, -1.4401, -0.8874],
 [-0.9492, -1.4018, -1.0030],
 [-0.8851, -1.4884, -1.0172],
 [-0.9620, -1.4861, -0.9375],
 [-0.9383, -1.4286, -0.9968],
 [-0.9473, -1.4407, -0.9797],
 [-1.0908, -1.3356, -0.9136],
 [-1.0898, -1.3512, -0.9043],
 [-0.9546, -1.4741, -0.9518],
 [-1.0250, -1.3957, -0.9325],
 [-0.9580, -1.4118, -0.9872],
 [-1.0212, -1.2985, -1.0027],
 [-1.0807, -1.3390, -0.9200],
 [-1.0069, -1.3805, -0.9592],
 [-0.9301, -1.4883, -0.9683],
 [-1.0115, -1.3251, -0.9927],
 [-0.9517, -1.4319, -0.9807],
 [-0.9659, -1.3798, -1.0004],
 [-0.8912, -1.5049, -1.0002],
 [-0.9809, -1.3245, -1.0241],
 [-0.9120, -1.4271, -1.0265],
 [-0.9053, -1.4309, -1.0314],
 [-0.9825, -1.3631, -0.9949],
 [-0.9912, -1.4685, -0.9198],
 [-0.9859, -1.4011, -0.9660],
 [-1.0054, -1.3767, -0.9631],
 [-0.9299, -1.3908, -1.0312],
 [-0.8797, -1.5794, -0.9702],
 [-1.0011, -1.3644, -0.9755],
 [-0.9722, -1.3923, -0.9855],
 [-0.9424, -1.5133, -0.9413],
 [-0.9472, -1.4304, -0.9863],
 [-0.9978, -1.4733, -0.9109],
 [-0.9955, -1.4294, -0.9391],
 [-1.0084, -1.3631, -0.9694],
 [-0.9013, -1.5595, -0.9578],
 [-0.9298, -1.4683, -0.9807],
 [-1.0009, -1.3300, -0.9997],
 [-0.9346, -1.5115, -0.9502]], device='cuda:0',
grad_fn=<LogSoftmaxBackward0>)
```

```
# no. of trainable parameters
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)

print(f'The model has {count_parameters(model):,} trainable parameters')
```

OUTPUT:

```
The model has 395,267 trainable parameters
```

```
# Adam optmizer
optimizer=torch.optim.Adam(model.parameters(),lr=0.0005)
```

```
# Understnding class distribution
keys=['0','1','2']
# set figure size
plt.figure(figsize=(5,5))
# plot bar chart
plt.bar(keys,class_counts)
# set title
plt.title('Class Distribution')
```

OUTPUT:

```
Text(0.5, 1.0, 'Class Distribution')
```


```
# library for array processing
import numpy as np

# computing the class weights
class_weights=compute_class_weight(class_weight='balanced',classes=np.unique(labels),y=labels)
print("Class Weights:",class_weights)
```

```
# Converting a list of class weights into a tensor
weights=torch.tensor(class_weights, dtype=torch.float)

# transferring weights to GPU
weights=weights.to(device)

# define the loss function
cross_entropy=nn.NLLLoss(weight=weights)
```

```
# Computing the loss
print(target)
#print(outputs)
loss=cross_entropy(outputs,target)
print('Loss: ',loss)
```

OUTPUT:

```
tensor([0, 1, 0, 0, 2, 1, 2, 0, 0, 2, 0, 0, 0, 1, 1, 2, 0, 0, 0, 0, 0,
0, 0, 0,
        0, 0, 0, 0, 2, 2, 0, 0, 0, 1, 0, 0, 0, 1, 2, 0, 1, 0, 0, 0, 0,
0, 0, 1,
        0, 2, 1, 0, 0, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0], device='cuda:0')
Loss:  tensor(1.0889, device='cuda:0', grad_fn=<NllLossBackward0>)
```

```
# Function for computing time in hh:mm:ss

def format_time(elapsed):

    elapsed_rounded=int(round(elapsed))

    # format intp hh:mm:ss
    return str(datetime.timedelta(seconds=elapsed_rounded))
```

Training Phase

```
# Defining a training function for the model:
def train():
    print('\n Training')
    # set the model on training phase- Dropout layers are activated
    model.train()
    # recording current time
    t0=time.time()
    # initialize the loss and accuracy to 0
    total_loss,total_accuracy=0,0
    # Create an empty list to save the model prediction
    total_preds=[]
    # for every batch
    for step, batch in enumerate(train_dataloader):
        #Progress update after every 40 batches
        if step % 40==0 and not step==0:
            elapsed=format_time(time.time()-t0)                 # Calculate elapsed time in minutes
            print(' Batch{:>5,} of {:>5,}. Elapsed: {:}.'.format(step,len(train_dataloader),elapsed)) # Print progress
        batch=tuple(t.to(device) for t in batch)            # push the batch to GPU
        # batch is a part of all the records in train_dataloader. It contains 3 pytorch tensors:
        # [0]: input ids
        # [1]: attention masks
        # [2]: labels
        sent_id,mask,labels=batch
  #Pytorch doesn't automatically clear previously calculated gradients, hence before performing a backward pass
        model.zero_grad()
        # Perform a forward pass. This returns the model predictions
        preds=model(sent_id,mask)
        # Compute the loss between actual and predicted values
        loss=cross_entropy(preds,labels)
        #Accumulate training loss over all the batches, so that we can calculate the average loss at the end
        # loss is a tensor containing a single value.
        # .itme() method just returns the Python value from the tensor
        total_loss=total_loss+loss.item()
        # Perform backward pass to calculate the gradients
        loss.backward()
        # During backward pass, information about parameter changes flows backwards, from the output to the hidden
layers to the input
        optimizer.step()
        # Update parameters and take a step using the computed gradient.
        # Here, the optimizer dictates the update rule = how the parameters are modified based on their gradients,
          # The model predictions are stored on GPU, so push it to CPU
        preds=preds.detach().cpu().numpy()
        # Accumulate model predicitons of each batch
        total_preds.append(preds)
  # Compute the training loss of an epoch
    avg_loss=total_loss/len(train_dataloader)
    # The prediction are in the form of (no. of batches, size of batch, no. of classes)
    # So we need to resahpe the predictions in the form of number of samples x number of classes
    total_preds=np.concatenate(total_preds, axis=0)
    return avg_loss,total_preds
```

Evaluation Phase

```
# define a function for evaluating the model

def evaluate():
    print("'n Evaluating....")

    # set the model on validation phase. Here dropout layers are deactivated
    model.eval()

    # record the current time
    t0=time.time()

    # initialize loss and accuracy to 0
    total_loss, total_accuracy=0,0

    # Create an empty list to save model predicitons
    total_preds=[]

    # for each batch

    for step, batch in enumerate(validation_dataloader):
        if step%40==0 and not step ==0:
            elapsed=format_time(time.time()-t0)
            print('    Batch {:>5,}    of    {:>5,}.        Elapsed: {:}.'.format(step, len(validation_dataloader), elapsed))

        batch=tuple(t.to(device) for t in batch)
        sent_id,mask,labels=batch

        #deactivate autograd
        with torch.no_grad():

            preds=model(sent_id,mask)
            loss=cross_entropy(preds,labels)
            total_loss=total_loss+loss.item()
            preds=preds.detach().cpu().numpy()
            total_preds.append(preds)

        avg_loss=total_loss/len(validation_dataloader)

        total_preds=np.concatenate(total_preds,axis=0)

        return avg_loss,total_preds
```

```
#define a function for evaluating the model
def evaluate():
    print("\nEvaluating.....")
    #set the model on training phase - Dropout layers are deactivated
    model.eval()
    #record the current time
    t0 = time.time()
    #initialize the loss and accuracy to 0
    total_loss, total_accuracy = 0, 0
    #Create a empty list to save the model predictions
    total_preds = []
    #for each batch
    for step,batch in enumerate(validation_dataloader):
        # Progress update every 40 batches.
        if step % 40 == 0 and not step == 0:
            # Calculate elapsed time in minutes.
            elapsed = format_time(time.time() - t0)
            # Report progress.
            print('    Batch {:>5,}    of    {:>5,}.        Elapsed: {:}.'.format(step, len(validation_dataloader), elapsed))
        #push the batch to gpu
        batch = tuple(t.to(device) for t in batch)
        #unpack the batch into separate variables
        # `batch` contains three pytorch tensors:
        #      [0]: input ids
        #      [1]: attention masks
        #      [2]: labels
        sent_id, mask, labels = batch
        #deactivates autograd
        with torch.no_grad():
            # Perform a forward pass. This returns the model predictions
            preds = model(sent_id, mask)
            #compute the validation loss between actual and predicted values
            loss = cross_entropy(preds,labels)
            # Accumulate the validation loss over all of the batches so that we can
            # calculate the average loss at the end. `loss` is a Tensor containing a
            # single value; the `.item()` function just returns the Python value
            # from the tensor.
            total_loss = total_loss + loss.item()
            #The model predictions are stored on GPU. So, push it to CPU
            preds=preds.detach().cpu().numpy()
            #Accumulate the model predictions of each batch
            total_preds.append(preds)
    #compute the validation loss of a epoch
    avg_loss = total_loss / len(validation_dataloader)
    #The predictions are in the form of (no. of batches, size of batch, no. of classes).
    #So, reshaping the predictions in form of (number of samples, no. of classes)
    total_preds    = np.concatenate(total_preds, axis=0)

    return avg_loss, total_preds
```

Train the model

```
# Assign the initial loss to infinite
best_valid_loss=float('inf')

# Create an empty list to store training and validation loss of each epoch
train_losses=[]
valid_losses=[]

epochs=5

#for each epoch repeat call the train() method
for epoch in range(epochs):
    print('\n ............epoch {:} / {:} .......'.format(epoch + 1, epochs))

    #train model
    train_loss,_ =train()

    #evaluate model
    valid_loss,_=evaluate()

    # save the best model
    if valid_loss<best_valid_loss:
        best_valid_loss=valid_loss
        torch.save(model.state_dict(),'Saved_weights.pt')

    # Accumulate training and validaion loss
    train_losses.append(train_loss)
    valid_losses.append(valid_loss)

    print(f'\nTraining Loss: {train_loss:.3f}')
    print(f'Validation Loss: {valid_loss:.3f}')

print("")
print("Training complete!")
```

OUTPUT:

```
............epoch 1 / 5 .......

 Training
 Batch   40 of   206. Elapsed: 0:00:04.
 Batch   80 of   206. Elapsed: 0:00:08.
 Batch  120 of   206. Elapsed: 0:00:13.
 Batch  160 of   206. Elapsed: 0:00:17.
 Batch  200 of   206. Elapsed: 0:00:21.

Evaluating.....

Training Loss: 0.716
Validation Loss: 0.746

 ............epoch 2 / 5 .......

 Training
 Batch   40 of   206. Elapsed: 0:00:04.
 Batch   80 of   206. Elapsed: 0:00:09.
 Batch  120 of   206. Elapsed: 0:00:13.
 Batch  160 of   206. Elapsed: 0:00:18.
 Batch  200 of   206. Elapsed: 0:00:22.

Evaluating.....

Training Loss: 0.716
Validation Loss: 0.701

 ............epoch 3 / 5 .......

 Training
 Batch   40 of   206. Elapsed: 0:00:05.
 Batch   80 of   206. Elapsed: 0:00:09.
 Batch  120 of   206. Elapsed: 0:00:14.
 Batch  160 of   206. Elapsed: 0:00:19.
 Batch  200 of   206. Elapsed: 0:00:24.

Evaluating.....

Training Loss: 0.709
Validation Loss: 0.656
```

```
 ............epoch 4 / 5 .......

 Training
 Batch   40 of   206. Elapsed: 0:00:05.
 Batch   80 of   206. Elapsed: 0:00:09.
 Batch  120 of   206. Elapsed: 0:00:14.
 Batch  160 of   206. Elapsed: 0:00:18.
 Batch  200 of   206. Elapsed: 0:00:23.

Evaluating.....

Training Loss: 0.705
Validation Loss: 0.645

 ............epoch 5 / 5 .......

 Training
 Batch   40 of   206. Elapsed: 0:00:04.
 Batch   80 of   206. Elapsed: 0:00:09.
 Batch  120 of   206. Elapsed: 0:00:13.
 Batch  160 of   206. Elapsed: 0:00:18.
 Batch  200 of   206. Elapsed: 0:00:22.

Evaluating.....

Training Loss: 0.705
Validation Loss: 0.640

Training complete!
```

Evaluate the model

```
# load weights of best model
path='Saved_weights.pt'
model.load_state_dict(torch.load(path))
```

OUTPUT:

<All keys matched successfully>

```
print(classification_report(y_true,y_pred))
```

```
# get the model prediction on the validation data
valid_loss, preds=evaluate()
# this returns 2 elements- Validation loss and prediction
print(valid_loss)
```

OUTPUT

Evaluating.....
0.6396074994750645

```
# Converting the log(probabilities) into class & then choosing index of maximum value as class
y_pred=np.argmax(preds,axis=1)

# actual labels
y_true=validation_labels
```

```
print(classification_report(y_true,y_pred))
```

OUTPUT:

	precision	recall	f1-score	support
0	0.93	0.66	0.78	918
1	0.49	0.70	0.58	310
2	0.56	0.89	0.69	236
accuracy			0.71	1464
macro avg	0.66	0.75	0.68	1464
weighted avg	0.78	0.71	0.72	1464

CLEANED DATA

```
In [1]: import pandas as pd

        # Load your CSV dataset
        df = pd.read_csv(r"C:\Users\sound\Downloads\Tweets.csv")

        print(df.head())
```

```
             tweet_id airline_sentiment  airline_sentiment_confidence  \
0  570306133677760513           neutral                        1.0000
1  570301130888122368          positive                        0.3486
2  570301083672813571           neutral                        0.6837
3  570301031407624196          negative                        1.0000
4  570300817074462722          negative                        1.0000

  negativereason  negativereason_confidence         airline  \
0            NaN                        NaN  Virgin America
1            NaN                     0.0000  Virgin America
2            NaN                        NaN  Virgin America
3     Bad Flight                     0.7033  Virgin America
4     Can't Tell                     1.0000  Virgin America

  airline_sentiment_gold        name negativereason_gold  retweet_count  \
0                    NaN     cairdin                 NaN              0
1                    NaN    jnardino                 NaN              0
2                    NaN  yvonnalynn                 NaN              0
3                    NaN    jnardino                 NaN              0
4                    NaN    jnardino                 NaN              0

                                                text tweet_coord  \
0                @VirginAmerica What @dhepburn said.         NaN
1  @VirginAmerica plus you've added commercials t...         NaN
2  @VirginAmerica I didn't today... Must mean I n...         NaN
3  @VirginAmerica it's really aggressive to blast...         NaN
4  @VirginAmerica and it's a really big bad thing...         NaN

               tweet_created tweet_location               user_timezone
0  2015-02-24 11:35:52 -0800            NaN  Eastern Time (US & Canada)
1  2015-02-24 11:15:59 -0800            NaN  Pacific Time (US & Canada)
2  2015-02-24 11:15:48 -0800      Lets Play  Central Time (US & Canada)
3  2015-02-24 11:15:36 -0800            NaN  Pacific Time (US & Canada)
4  2015-02-24 11:14:45 -0800            NaN  Pacific Time (US & Canada)
```

```
In [2]: df = df.drop_duplicates()
        # 2. Remove rows with missing values
        df = df.dropna()

        df = df.apply(lambda x: x.str.strip() if x.dtype == "object" else x)

        df.to_csv('cleaned_file.csv', index=False)
        print(df)
```



```
                tweet_id airline_sentiment  airline_sentiment_confidence  \
4206  567778009013178368          negative                        1.0000
9536  569887533267611648          negative                        0.8563

        negativereason  negativereason_confidence     airline  \
4206  Cancelled Flight                     1.0000      United
9536       Late Flight                     0.5938  US Airways

     airline_sentiment_gold             name negativereason_gold  \
4206               negative    realmikesmith    Cancelled Flight
9536               negative  ConstanceSCHERE         Late Flight

      retweet_count                                               text  \
4206              0  @united So what do you offer now that my fligh...
9536              0  @USAirways Seriously doubt that as I am still ...

                       tweet_coord              tweet_created tweet_location  \
4206  [26.37852293, -81.78472152]  2015-02-17 12:10:00 -0800        Chicago
9536   [39.8805621, -75.23893393]  2015-02-23 07:52:30 -0800     Boston, MA

                   user_timezone
4206  Eastern Time (US & Canada)
9536      Atlantic Time (Canada)
```

```
In [ ]:
```

CONCLUSION

BERT and RoBERTa are two leading models in the field of natural language processing (NLP), frequently used for sentiment analysis tasks. BERT, known as Bidirectional Encoder Representations from Transformers, possesses a strong advantage in its ability to capture contextual information effectively. This contextual understanding allows BERT to excel in discerning the nuances of sentiment in a given text, particularly in cases where sentiment depends on the surrounding context. However, it comes with the drawback of being computationally intensive, which can be a limitation for real-time or resource-constrained applications. Nevertheless, it offers flexibility through fine-tuning for specific sentiment analysis tasks.

On the other hand, RoBERTa, an optimized variant of BERT, has undergone various training enhancements, making it more efficient and robust. It serves as a strong baseline model for sentiment analysis, providing competitive results even without extensive fine-tuning. RoBERTa is designed to be computationally efficient while delivering high performance, making it a preferred choice for tasks where efficiency and resource utilization are critical. This efficiency can be especially advantageous when dealing with constrained computational resources or when a strong, pre-trained model is needed with minimal fine-tuning.

In conclusion, the choice between BERT and RoBERTa for sentiment analysis depends on the specific requirements of your project. BERT is ideal when fine-tuning with domain-specific data is feasible and maximum accuracy is essential. In contrast, RoBERTa is a strong contender when you seek a high-performing model without extensive fine-tuning or when computational efficiency is a priority, ensuring that you can make the best choice to meet your project's specific needs.