```
!pip install opendatasets
```

```
Collecting opendatasets
  Downloading opendatasets-0.1.22-py3-none-any.whl (15 kB)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (from opendatasets) (4.66.1)
Requirement already satisfied: kaggle in
/usr/local/lib/python3.10/dist-packages (from opendatasets) (1.5.16)
Requirement already satisfied: click in
/usr/local/lib/python3.10/dist-packages (from opendatasets) (8.1.7)
Requirement already satisfied: six>=1.10 in
/usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets)
(1.16.0)
Requirement already satisfied: certifi in
/usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets)
(2023.11.17)
Requirement already satisfied: python-dateutil in
/usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets)
(2.8.2)
Requirement already satisfied: requests in
/usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets)
(2.31.0)
Requirement already satisfied: python-slugify in
/usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets)
(8.0.1)
Requirement already satisfied: urllib3 in
/usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets)
(2.0.7)
Requirement already satisfied: bleach in
/usr/local/lib/python3.10/dist-packages (from kaggle->opendatasets)
(6.1.0)
Requirement already satisfied: webencodings in
/usr/local/lib/python3.10/dist-packages (from bleach->kaggle-
>opendatasets) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in
/usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle-
>opendatasets) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->kaggle-
>opendatasets) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests->kaggle-
>opendatasets) (3.6)
Installing collected packages: opendatasets
Successfully installed opendatasets-0.1.22
```

```
import opendatasets as od
import numpy as np
import pandas as pd
import seaborn as sns
```

```
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
sns.set(color_codes=True)
```

```
od.download("https://www.kaggle.com/datasets/kazanova/sentiment140")
```

```
Please provide your Kaggle credentials to download this dataset. Learn more: http://bit.ly/kaggle-creds
Your Kaggle username: mithilkatkar
Your Kaggle Key: ·····
Downloading sentiment140.zip to ./sentiment140

100%|██████████| 80.9M/80.9M [00:00<00:00, 187MB/s]
```

```
df = pd.read_csv('/content/sentiment140/training.1600000.processed.noemoticon.csv', encoding='latin-1', header = None)
df.columns=['Sentiment', 'id', 'Date', 'Query', 'User', 'Tweet']
df = df.drop(columns=['id', 'Date', 'Query', 'User'], axis=1)
```

```
df.head()
```

	Sentiment	Tweet
0	0	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	is upset that he can't update his Facebook by ...
2	0	@Kenichan I dived many times for the ball. Man...
3	0	my whole body feels itchy and like its on fire
4	0	@nationwideclass no, it's not behaving at all....

```
df['Sentiment'] = df.Sentiment.replace(4,1)
```

```
import re

hashtags = re.compile(r"^#\S+|\s#\S+")
mentions = re.compile(r"^@\S+|\s@\S+")
urls = re.compile(r"https?://\S+")

def process_text(text):
    text = re.sub(r'http\S+', '', text)
    text = hashtags.sub(' hashtag', text)
    text = mentions.sub(' entity', text)
    return text.strip().lower()

df['Tweet'] = df.Tweet.apply(process_text)
```

```
df.head()
```

```
   Sentiment                                              Tweet
0          0  entity  - awww, that's a bummer.  you shoulda ...
1          0  is upset that he can't update his facebook by ...
2          0  entity i dived many times for the ball. manage...
3          0     my whole body feels itchy and like its on fire
4          0  entity no, it's not behaving at all. i'm mad. ...
```

```
labels = df.Sentiment.values
text = df.Tweet.values
```

```
from transformers import
BertTokenizer,BertForSequenceClassification,AdamW
```

```
tokenizer = BertTokenizer.from_pretrained('bert-base-
uncased',do_lower_case = True)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/
_token.py:88: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
access public models or datasets.
  warnings.warn(
```


```
{"model_id":"b3348c12dcef4df4b9ba1eb8918f18bc","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"46dd10d59d05480f8e8cdc3536fff89b","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"126e871887df499295cea14a70eb2509","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"dcf08089b90f4a4ca6ad1adcf108e661","version_major":2,"vers
ion_minor":0}
```


```
input_ids = []
attention_mask = []
for i in text:
    encoded_data = tokenizer.encode_plus(
    i,
    add_special_tokens=True,
    max_length=64,
    pad_to_max_length = True,
    return_attention_mask= True,
    return_tensors='pt')
    input_ids.append(encoded_data['input_ids'])
    attention_mask.append(encoded_data['attention_mask'])
```

```
input_ids = torch.cat(input_ids,dim=0)
attention_mask = torch.cat(attention_mask,dim=0)
labels = torch.tensor(labels)
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly truncate examples to max length. Defaulting to 'longest_first' truncation strategy. If you encode pairs of sequences (GLUE-style) with the tokenizer you can select this strategy more precisely by providing a specific strategy to `truncation`.
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:2614: FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a future version, use `padding=True` or `padding='longest'` to pad to the longest sequence in the batch, or use `padding='max_length'` to pad to a max length. In this case, you can give a specific length with `max_length` (e.g. `max_length=45`) or leave max_length to None to pad to the maximal input size of the model (e.g. 512 for Bert).
  warnings.warn(
```

```
from torch.utils.data import
DataLoader,SequentialSampler,RandomSampler,TensorDataset,random_split

dataset = TensorDataset(input_ids,attention_mask,labels)
train_size = int(0.8*len(dataset))
val_size = len(dataset) - train_size

train_dataset,val_dataset = random_split(dataset,
[train_size,val_size])

print('Training Size - ',train_size)
print('Validation Size - ',val_size)
```

```
Training Size -  1280000
Validation Size -  320000
```

```
train_dl = DataLoader(train_dataset,sampler =
RandomSampler(train_dataset),
                      batch_size = 50)
val_dl = DataLoader(val_dataset,sampler =
SequentialSampler(val_dataset),
                      batch_size = 50)

len(train_dl),len(val_dl)
```

```
(25600, 6400)
```

```
model = BertForSequenceClassification.from_pretrained(
'bert-base-uncased',
num_labels = 2,
```

```
output_attentions = False,
output_hidden_states = False)
```

```
{"model_id":"f1b846b38d8e43ae96ea452628fe3a4d","version_major":2,"vers
ion_minor":0}
```

```
Some weights of BertForSequenceClassification were not initialized
from the model checkpoint at bert-base-uncased and are newly
initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
```

```
import random

seed_val = 17
random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

print(device)
```

```
cuda
```

```
optimizer = AdamW(model.parameters(),lr = 2e-5,eps=1e-8)
```

```
/usr/local/lib/python3.10/dist-packages/transformers/
optimization.py:411: FutureWarning: This implementation of AdamW is
deprecated and will be removed in a future version. Use the PyTorch
implementation torch.optim.AdamW instead, or set
`no_deprecation_warning=True` to disable this warning
  warnings.warn(
```

```
from transformers import get_linear_schedule_with_warmup
epochs = 1
total_steps = len(train_dl)*epochs
scheduler =
get_linear_schedule_with_warmup(optimizer,num_warmup_steps=0,

num_training_steps=total_steps)

def accuracy(preds,labels):
    pred_flat = np.argmax(preds,axis=1).flatten()
    label_flat = labels.flatten()
    return np.sum(pred_flat==label_flat)/len(label_flat)

def evaluate(dataloader_test):
    model.eval()
    loss_val_total = 0
```

```
    predictions,true_vals = [],[]
    for batch in dataloader_test:
        batch = tuple(b.to(device) for b in batch)
        inputs = {
            'input_ids':batch[0],
            'attention_mask': batch[1],
            'labels': batch[2]
        }
        with torch.no_grad():
            outputs = model(**inputs)
        loss = outputs[0]
        logits = outputs[1]
        loss_val_total += loss.item()
        logits = logits.detach().cpu().numpy()
        label_ids = inputs['labels'].cpu().numpy()
        predictions.append(logits)
        true_vals.append(label_ids)
    loss_val_avg = loss_val_total / len(dataloader_test)
    predictions = np.concatenate(predictions,axis=0)
    true_vals = np.concatenate(true_vals,axis=0)
    return loss_val_avg,predictions,true_vals

from tqdm.notebook import tqdm
from torch.cuda.amp import autocast
torch.cuda.empty_cache()
for epoch in tqdm(range(1, epochs+1)):

    model.train()

    loss_train_total = 0

    progress_bar = tqdm(train_dl, desc='Epoch {:1d}'.format(epoch),
leave=False, disable=False)
    for batch in progress_bar:

        model.zero_grad()

        batch = tuple(b.to(device) for b in batch)

        with autocast():
            inputs = {'input_ids': batch[0], 'attention_mask':
batch[1], 'labels': batch[2]}
            outputs = model(**inputs)
            loss = outputs[0]

        loss_train_total += loss.item()
        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
```

```
        optimizer.step()
        scheduler.step()

        progress_bar.set_postfix({'training_loss':
'{:.3f}'.format(loss.item()/len(batch))})


    tqdm.write(f'\nEpoch {epoch}')

    loss_train_avg = loss_train_total/len(train_dl)
    tqdm.write(f'Training loss: {loss_train_avg}')

    val_loss, predictions, true_vals = evaluate(val_dl)
    val_acc = accuracy(predictions, true_vals)
    tqdm.write(f'Validation loss: {val_loss}')
    tqdm.write(f'Accuracy: {val_acc}')
```


```
{"model_id":"11c39417c65f4a328a14181084924059","version_major":2,"vers
ion_minor":0}

{"model_id":"7e248650b79b46b0bbf1fca0cc120b8b","version_major":2,"vers
ion_minor":0}
```


```
Epoch 1
Training loss: 0.32496722998592303
Validation loss: 0.3000223604729399
Accuracy: 0.872825
```

```
output_dir = './'
model_to_save = model.module if hasattr(model, 'module') else model
model_to_save.save_pretrained(output_dir)
tokenizer.save_pretrained(output_dir)
```

```
('./tokenizer_config.json',
 './special_tokens_map.json',
 './vocab.txt',
 './added_tokens.json')
```

```
from transformers import BertTokenizer,BertForSequenceClassification
import torch
# Load the BERT tokenizer.
print('Loading BERT tokenizer...')
output_dir = './'
tokenizer = BertTokenizer.from_pretrained(output_dir)
model_loaded =
BertForSequenceClassification.from_pretrained(output_dir)
```

```
Loading BERT tokenizer...
```

```
def Sentiment(sent):
    output_dir = './'
```

```
    tokenizer = BertTokenizer.from_pretrained(output_dir)
    model_loaded =
BertForSequenceClassification.from_pretrained(output_dir)
    encoded_dict = tokenizer.encode_plus(
                        sent,
                        add_special_tokens = True,
                        max_length = 64,
                        pad_to_max_length = True,
                        return_attention_mask = True,
                        return_tensors = 'pt',
                   )

    input_id = encoded_dict['input_ids']

    attention_mask = encoded_dict['attention_mask']
    input_id = torch.LongTensor(input_id)
    attention_mask = torch.LongTensor(attention_mask)

    device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
    model_loaded = model_loaded.to(device)
    input_id = input_id.to(device)
    attention_mask = attention_mask.to(device)

    with torch.no_grad():
        outputs = model_loaded(input_id, token_type_ids=None,
attention_mask=attention_mask)

    logits = outputs[0]
    index = logits.argmax()
    return index

ans = Sentiment('i want to die')
```

```
Truncation was not explicitly activated but `max_length` is provided a
specific value, please use `truncation=True` to explicitly truncate
examples to max length. Defaulting to 'longest_first' truncation
strategy. If you encode pairs of sequences (GLUE-style) with the
tokenizer you can select this strategy more precisely by providing a
specific strategy to `truncation`.
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_util
s_base.py:2614: FutureWarning: The `pad_to_max_length` argument is
deprecated and will be removed in a future version, use `padding=True`
or `padding='longest'` to pad to the longest sequence in the batch, or
use `padding='max_length'` to pad to a max length. In this case, you
can give a specific length with `max_length` (e.g. `max_length=45`) or
leave max_length to None to pad to the maximal input size of the model
(e.g. 512 for Bert).
  warnings.warn(
```

```
if ans == 1:
    print("Positive")
else:
    print("Negative")
```

```
Negative
```

```
ans1 = Sentiment('I am so happy today')
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly truncate examples to max length. Defaulting to 'longest_first' truncation strategy. If you encode pairs of sequences (GLUE-style) with the tokenizer you can select this strategy more precisely by providing a specific strategy to `truncation`.
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:2614: FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a future version, use `padding=True` or `padding='longest'` to pad to the longest sequence in the batch, or use `padding='max_length'` to pad to a max length. In this case, you can give a specific length with `max_length` (e.g. `max_length=45`) or leave max_length to None to pad to the maximal input size of the model (e.g. 512 for Bert).
  warnings.warn(
```

```
if ans1 == 1:
    print("Positive")
else:
    print("Negative")
```

```
Positive
```

```
ans2 = Sentiment('Today is the best day of my life')
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly truncate examples to max length. Defaulting to 'longest_first' truncation strategy. If you encode pairs of sequences (GLUE-style) with the tokenizer you can select this strategy more precisely by providing a specific strategy to `truncation`.
```

```
if ans2 == 1:
    print("Positive")
else:
    print("Negative")
```

```
Positive
```