Twitter Sentiment Analysis with Deep Learning using BERT**

What is BERT?

BERT is a large-scale transformer-based Language Model that can be finetuned for a variety of tasks.

For more information, the original paper can be found here (https://arxiv.org/abs/1810.04805).

HuggingFace documentation (https://huggingface.co/transformers/model_doc/bert.html)

1: Exploratory Data Analysis and Preprocessing

```
!pip install torch #no change
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.1.0+cu118)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch) (2.1.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch) (2.1.3)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch) (1.3.0)
```

```
import torch #no change
from tqdm.notebook import tqdm #no change
import numpy as np #no change
import pandas as pd #no change
import os
os.environ['TORCH_USE_CUDA_DSA'] = '1'
```

```
df = pd.read_csv('/content/training.1600000.processed.noemoticon.csv',encoding='latin1')
```

```
df.head()
```

	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer. You shoulda got David Carr of Third Day to do it. ;D
0	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
1	0	1467810917	Mon Apr 06 22:19:53	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man

Note :since it is not possible to acces all the data from google colab we have identified the data range of 10000 tweets with both equal positive and negative polarity

```
df=df[795000:805000]
```

```
df=df.reset_index()
```

```
df.rename(columns={'0':'target',
                   '1467810369':'ids',
                   'Mon Apr 06 22:19:45 PDT 2009':'date',
                   'NO_QUERY':'flag',
                   '_TheSpecialOne_':'user',
                   "@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer. You shoulda got David Carr of Third Day to do it. ;
          inplace=True )
```

```
df=df[['target', "@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer.  You shoulda got David Carr of Third Day to do it. ;D"]
```

```
df.columns = ['DV','IV']
```

```
df.head()
```

	DV	IV
0	0	Blah 5am still up daang I got deep problems
1	0	@jenspeedy I would suggest avoiding 360 Living...
2	0	@alexbroun I didn't convince myself I was fat ...
3	0	@spotzle @jstarrh check on sunscreen, snacks, ...
4	0	im sitting alone at TTE myself without my two ...

```
df['DV'] =df['DV'].replace(0,'negative_polarity')
```

```
df.head()
```

	DV	IV
0	negative_polarity	Blah 5am still up daang I got deep problems
1	negative_polarity	@jenspeedy I would suggest avoiding 360 Living...
2	negative_polarity	@alexbroun I didn't convince myself I was fat ...
3	negative_polarity	@spotzle @jstarrh check on sunscreen, snacks, ...
4	negative_polarity	im sitting alone at TTE myself without my two ...

```
df['DV']=df['DV'].replace(4,'positive_polarity')
```

```
df.tail()
```

	DV	IV
9995	positive_polarity	@ickleoriental hahahha.. U obviously don't hv ...
9996	positive_polarity	@juliekoh It's an internet term, but it's spil...
9997	positive_polarity	new day.... NEW TRACK!!!!
9998	positive_polarity	@foodieguide Okay we need to have a competitio...
9999	positive_polarity	@PerfectElement noooooooo i wish, I just saw i...

```
set(df.DV)
```

```
{'negative_polarity', 'positive_polarity'}
```

```
possible_labels = df.DV.unique()
```

```
possible_labels
```

```
label_dict = {}
for index, possible_label in enumerate(possible_labels):
    label_dict[possible_label] = index
```

```
df['DV']=df['DV'].map(label_dict)
```

```
df.DV.value_counts()
```

```
1    5001
0    4999
Name: DV, dtype: int64
```

```
df.DV.unique()
```

```
array([0, 1])
```

Classes are imbalanced as visible

2: Training/Validation Split

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_val, y_train, y_val = train_test_split(df.index.values,
                                                  df.DV.values,
                                                  test_size=0.15,
                                                  random_state=42,
                                                  stratify=df.DV.values)
```

```
X_train
```

```
X_val
```

```
array([2261,  899, 4029, ..., 4946,  769, 6377])
```

```
df.head()
```

	DV	IV
0	0	Blah 5am still up daang I got deep problems
1	0	@jenspeedy I would suggest avoiding 360 Living...
2	0	@alexbroun I didn't convince myself I was fat ...
3	0	@spotzle @jstarrh check on sunscreen, snacks, ...
4	0	im sitting alone at TTE myself without my two ...

```
len(df)
```

```
10000
```

```
df.shape
```

```
(10000, 2)
```

```
df.shape[0]
```

```
10000
```

```
df['data_type'] = ['not_set']*df.shape[0]
```

```
df.head()
```

	DV	IV	data_type
0	0	Blah 5am still up daang I got deep problems	not_set
1	0	@jenspeedy I would suggest avoiding 360 Living...	not_set
2	0	@alexbroun I didn't convince myself I was fat ...	not_set
3	0	@spotzle @jstarrh check on sunscreen, snacks, ...	not_set
4	0	im sitting alone at TTE myself without my two ...	not_set

```
df.loc[X_train, 'data_type'] = 'train'
df.loc[X_val, 'data_type'] = 'val'
```

```
df
```

```
df[df['data_type']=="val"]
```

	DV	IV	data_type
4	0	im sitting alone at TTE myself without my two ...	val
5	0	@Julie90210 It took me three attempts but I go...	val
9	0	Bleurgh...feeling rough	val
16	0	stupid rain just woke me up. already sense a b...	val
34	0	pls someone help me to put my eyes away from t...	val
...	...	...	...
9975	1	@blurtit thank you for answering my question.	val
9983	1	had a great time yesterday Thanks everyone fo...	val
9992	1	@clairelouisef lol I try. Maybe not hard enoug...	val
9994	1	@lbran, thanks for sending us the package - go...	val
9995	1	@ickleoriental hahahha.. U obviously don't hv ...	val

1500 rows × 3 columns

```
df.groupby(['DV', 'data_type']).count()
```

DV	data_type	IV
0	**train**	4249
	val	750
1	**train**	4251
	val	750

3. Loading Tokenizer and Encoding our Data

```
!pip install transformers
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.35.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.13.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.19.4)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2023.6.3)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.31.0)
Requirement already satisfied: tokenizers<0.19,>=0.14 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.15.0)
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.4.1)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->tran
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2023.11
```

```
from transformers import BertTokenizer
from torch.utils.data import TensorDataset
```

```
tokenizer = BertTokenizer.from_pretrained(
    'bert-large-uncased', #bert-base-uncased using small bert model for simple data , bert-large-uncased IT WILL USE LARGE MODEL fo lar
    do_lower_case=True
)
```

```
tokenizer_config.json: 100%  28.0/28.0 [00:00<00:00, 1.55kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 2.31MB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 2.21MB/s]
config.json: 100%  571/571 [00:00<00:00, 28.0kB/s]
```

```
encoded_data_train = tokenizer.batch_encode_plus(

    df[df.data_type=='train'].IV.values,
    add_special_tokens=True,
    return_attention_mask=True,
    pad_to_max_length=True,

    return_tensors='pt'
)

encoded_data_val = tokenizer.batch_encode_plus(
    df[df.data_type=='val'].IV.values,
    add_special_tokens=True,
    return_attention_mask=True,
    pad_to_max_length=True,

    return_tensors='pt'
)

input_ids_train = encoded_data_train['input_ids']
attention_masks_train = encoded_data_train['attention_mask']
labels_train = torch.tensor(df[df.data_type=='train'].DV.values)

input_ids_val = encoded_data_val['input_ids']
attention_masks_val = encoded_data_val['attention_mask']
labels_val = torch.tensor(df[df.data_type=='val'].DV.values)
```

```
input_ids_train
```

```
tensor([[  101, 27984,  1019,  ...,     0,     0,     0],
        [  101,  1030, 25093,  ...,     0,     0,     0],
        [  101,  1030,  4074,  ...,     0,     0,     0],
        ...,
        [  101,  2047,  2154,  ...,     0,     0,     0],
        [  101,  1030,  2833,  ...,     0,     0,     0],
        [  101,  1030,  3819,  ...,     0,     0,     0]])
```

```
attention_masks_train
```

```
tensor([[1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        ...,
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0]])
```

```
dataset_train = TensorDataset(input_ids_train,
                              attention_masks_train,
                              labels_train)

dataset_val = TensorDataset(input_ids_val,
                            attention_masks_val,
                           labels_val)
```

```
len(dataset_train)
```

```
8500
```

```
dataset_train.tensors
```

```
(tensor([[  101, 27984,  1019,  ...,     0,     0,     0],
         [  101,  1030, 25093,  ...,     0,     0,     0],
```

```
        [  101,  1030,  4074,  ...,     0,     0,     0],
        ...,
        [  101,  2047,  2154,  ...,     0,     0,     0],
        [  101,  1030,  2833,  ...,     0,     0,     0],
        [  101,  1030,  3819,  ...,     0,     0,     0]]),
 tensor([[1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        ...,
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0]]),
 tensor([0, 0, 0,  ..., 1, 1, 1]))
```

4. Setting up BERT Pretrained Model

```
from transformers import BertForSequenceClassification
```

```
model = BertForSequenceClassification.from_pretrained(
                                      'bert-large-uncased',
                                      num_labels =len(label_dict),
                                      output_attentions = False,
                                      output_hidden_states = False
                                     )
```

```
model.safetensors:                                                1.34G/1.34G [00:59<00:00,
100%                                                              17.1MB/s]
Some weights of BertForSequenceClassification were not initialized from the model che
```

5. Creating Data Loaders

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

```
dataset_train
```

```
<torch.utils.data.dataset.TensorDataset at 0x7ba46d595f60>
```

```
batch_size = 4

dataloader_train = DataLoader(
    dataset_train,
    sampler=RandomSampler(dataset_train),
    batch_size=batch_size
)

dataloader_val = DataLoader(
    dataset_val,
    sampler=RandomSampler(dataset_val),
    batch_size=32
)
```

6. Setting Up Optimizer and Scheduler

```
from transformers import AdamW, get_linear_schedule_with_warmup
```

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

```
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:411: FutureWarning: This implementation of AdamW is deprecated
  warnings.warn(
```

```
epochs = 5

scheduler = get_linear_schedule_with_warmup(
    optimizer,
    num_warmup_steps=0,
    num_training_steps = len(dataloader_train)*epochs
)
```

7. Defining our Performance Metrics

```
import numpy as np
from sklearn.metrics import f1_score


def f1_score_func(preds, labels):
    preds_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    return f1_score(labels_flat, preds_flat, average = 'weighted')


def accuracy_per_class(preds, labels):
    label_dict_inverse = {v: k for k, v in label_dict.items()}

    preds_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()

    for label in np.unique(labels_flat):
        y_preds = preds_flat[labels_flat==label]
        y_true = labels_flat[labels_flat==label]
        print(f'Class: {label_dict_inverse[label]}')
        print(f'Accuracy:{len(y_preds[y_preds==label])}/{len(y_true)}\n')
```

8. Creating our Training Loop

```
CUDA_LAUNCH_BLOCKING=1
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
```

```
def evaluate(dataloader_val):

    model.eval()

    loss_val_total = 0
    predictions, true_vals = [], []

    for batch in tqdm(dataloader_val):

        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_val)

    predictions = np.concatenate(predictions, axis=0)
    true_vals = np.concatenate(true_vals, axis=0)

    return loss_val_avg, predictions, true_vals


for epoch in tqdm(range(1, epochs+1)):
    model.train() #forward propagation
    loss_train_total = 0

    progress_bar = tqdm(dataloader_train,
                        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)
        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() #backwardprop

        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))})

    torch.save(model, f'BERT_ft_Epoch_{epoch}.model')

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

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

    val_loss, predictions, true_vals = evaluate(dataloader_val)
    val_f1 = f1_score_func(predictions, true_vals)
    tqdm.write(f'Validation loss: {val_loss}')
    tqdm.write(f'F1 Score (weighted): {val_f1}')
```

```
100%                                          5/5 [53:46<00:00, 647.30s/it]

Epoch 1
Training loss: 0.5746917913056472
100%                                          47/47 [00:15<00:00, 3.02it/s]
Validation loss: 0.4852697917438568
F1 Score (weighted): 0.8470461199363247

Epoch 2
Training loss: 0.38468762175330673
100%                                          47/47 [00:15<00:00, 3.01it/s]
Validation loss: 0.6941861646606567
F1 Score (weighted): 0.8506217868123934

Epoch 3
Training loss: 0.1890620134836679
100%                                          47/47 [00:15<00:00, 3.04it/s]
Validation loss: 0.8072603604895003
F1 Score (weighted): 0.8586304760685401

Epoch 4
Training loss: 0.08135885391739259
100%                                          47/47 [00:15<00:00, 3.03it/s]
Validation loss: 1.0361439346316013
F1 Score (weighted): 0.8613175543528508

Epoch 5
Training loss: 0.03387114531210853
```

EVALUATION

```
import torch
```

```
tweet = "this is the best day in my life :)"
```

```
from transformers import BertTokenizer
```

```
tokenizer = BertTokenizer.from_pretrained(
    'bert-base-uncased', #bert-base-uncased using small bert model for simple data , bert-large-uncased fo large data
    do_lower_case=True
)
```

```
tokenizer_config.json:                                    28.0/28.0 [00:00<00:00,
100%                                                      1.63kB/s]
vocab.txt: 100%                                   232k/232k [00:00<00:00, 6.03MB/s]
tokenizer.json: 100%                                  466k/466k [00:00<00:00, 13.9MB/s]
```

```
device = torch.device('cpu')
```

```
print(device)
```

```
cpu
```

```
encoded_headline = tokenizer(tweet, return_tensors = 'pt')
```

```
encoded_headline
```

```
input_ids = encoded_headline['input_ids'].to(device)
attention_msk = encoded_headline['attention_mask'].to(device)
```

```
path = '/content/BERT_ft_Epoch_5.model'
model = torch.load(path, map_location = torch.device('cpu'))
```

```
model
```

```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 1024, padding_idx=0)
      (position_embeddings): Embedding(512, 1024)
      (token_type_embeddings): Embedding(2, 1024)
      (LayerNorm): LayerNorm((1024,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-23): 24 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=1024, out_features=1024, bias=True)
              (key): Linear(in_features=1024, out_features=1024, bias=True)
              (value): Linear(in_features=1024, out_features=1024, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=1024, out_features=1024, bias=True)
              (LayerNorm): LayerNorm((1024,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=1024, out_features=4096, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=4096, out_features=1024, bias=True)
            (LayerNorm): LayerNorm((1024,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=1024, out_features=1024, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=1024, out_features=2, bias=True)
)
```

```
model_output = model(input_ids,attention_msk)
```

```
model_output
```

```
SequenceClassifierOutput(loss=None, logits=tensor([[-5.9016,  5.4237]], grad_fn=<AddmmBackward0>), hidden_states=None,
attentions=None)
```

```
model_output_tensor = torch.tensor(model_output.logits)
```

```
<ipython-input-81-d2b33ccd89fe>:1: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.clone().deta
  model_output_tensor = torch.tensor(model_output.logits)
```

```
model_output_tensor
```

```
tensor([[-5.9016,  5.4237]])
```

```
model_output_tensor_categoryIndex = int(torch.argmax(model_output_tensor))
```

```
model_output_tensor_categoryIndex
```

```
1
```

```
# classes = {0: 'Non-biased', 1: 'Biased'}
label_dict
```

```
{'negative_polarity': 0, 'positive_polarity': 1}
```

```
swapped_dict = {v: k for k, v in label_dict.items()}
```