NLP Exercise: Experiementing with the state-of-the-art NLP pre-trained models BERT

The goal of this exercise is to understand the mechanism of BERT, the state - of - art transformer model in dealing with textual data. In this notebook, I will be following the tutorial suggested from the huggingface (link1 and link2) on canvas. The dataset of imdb will be used.

The reason for choosing the distilBert model is to train faster on data values, for experiment I'll be using around 500 points from test and train set from the imdb dataset.

1. Importing the required libraries

Firstly, I'll be importing the required libraries and installing the libraries.

```
#Installing the datasets library from the transformers
! pip install transformers datasets
```

```
Collecting datasets
  Downloading datasets-2.6.1-py3-none-any.whl (441 kB)
     |████████████████████████████████| 441 kB 54.7 MB/s
Collecting tokenizers!=0.11.3,<0.14,>=0.11.1
  Downloading tokenizers-0.13.1-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (7.6 MB)
     |████████████████████████████████| 7.6 MB 56.0 MB/s
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.7/dist-packages (from transformers) (2022
Collecting huggingface-hub<1.0,>=0.10.0
  Downloading huggingface_hub-0.10.1-py3-none-any.whl (163 kB)
     |████████████████████████████████| 163 kB 58.5 MB/s
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packages (from transformers) (3.8.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-packages (from transformers) (21.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.7/dist-packages (from transformers) (4.64.1)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages (from transformers) (1.21.6)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-packages (from transformers) (4.1

Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.7/dist-packages (from transformers) (6.0)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from transformers) (2.23.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.7/dist-packages (from huggingfac
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging>=2
```

```
Requirement already satisfied: pyarrow>=6.0.0 in /usr/local/lib/python3.7/dist-packages (from datasets) (6.0.1)
Collecting multiprocess
  Downloading multiprocess-0.70.13-py37-none-any.whl (115 kB)
     |████████████████████████████████| 115 kB 43.4 MB/s
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from datasets) (1.3.5)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.7/dist-packages (from datasets) (3.8.3)
Requirement already satisfied: dill<0.3.6 in /usr/local/lib/python3.7/dist-packages (from datasets) (0.3.5.1)
Collecting xxhash
  Downloading xxhash-3.0.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (212 kB)
     |████████████████████████████████| 212 kB 74.2 MB/s
Collecting responses<0.19
  Downloading responses-0.18.0-py3-none-any.whl (38 kB)
Requirement already satisfied: fsspec[http]>=2021.11.1 in /usr/local/lib/python3.7/dist-packages (from datasets) (20
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.7/dist-packages (from aiohttp->datasets) (22.
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.7/dist-packages (from aiohttp->datasets
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in /usr/local/lib/python3.7/dist-packages (from aiohttp->
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.7/dist-packages (from aiohttp->datasets) (1.
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.7/dist-packages (from aiohttp->datasets)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.7/dist-packages (from aiohttp->datasets) (
Requirement already satisfied: asynctest==0.13.0 in /usr/local/lib/python3.7/dist-packages (from aiohttp->datasets)
Requirement already satisfied: charset-normalizer<3.0,>=2.0 in /usr/local/lib/python3.7/dist-packages (from aiohttp-
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->transfor
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->transform
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->transformers)
Collecting urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1
  Downloading urllib3-1.25.11-py2.py3-none-any.whl (127 kB)
     |████████████████████████████████| 127 kB 72.4 MB/s
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata->transfo
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas->datasets) (2022.
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas->datase
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pand
Installing collected packages: urllib3, xxhash, tokenizers, responses, multiprocess, huggingface-hub, transformers,
  Attempting uninstall: urllib3
    Found existing installation: urllib3 1.24.3
    Uninstalling urllib3-1.24.3:
      Successfully uninstalled urllib3-1.24.3
Successfully installed datasets-2.6.1 huggingface-hub-0.10.1 multiprocess-0.70.13 responses-0.18.0 tokenizers-0.13.1
```

```
#data manipulations
import pandas as pd
```

```
import numpy as np

#Bert modelling and fine-tuning
from datasets import load_dataset
import tensorflow as tf
from transformers import TFAutoModelForSequenceClassification
from transformers import AutoTokenizer
from transformers import DefaultDataCollator
import tensorflow_datasets as tfds

#for inference
from transformers import pipeline

#plotting
import matplotlib.pyplot as plt

%matplotlib inline


#loading the imdb dataset
df = load_dataset("imdb")
```

```
Downloading builder script: 100%  4.31k/4.31k [00:00<00:00, 78.8kB/s]
Downloading metadata: 100%  2.17k/2.17k [00:00<00:00, 69.2kB/s]
Downloading readme: 100%  7.16k/7.16k [00:00<00:00, 224kB/s]
Downloading and preparing dataset imdb/plain_text (download: 80.23 MiB, generated: 127.02 MiB, post-prc
Downloading data: 100%  84.1M/84.1M [00:04<00:00, 26.2MB/s]
Dataset imdb downloaded and prepared to /root/.cache/huggingface/datasets/imdb/plain_text/1.0.0/2fdd8b9
100%  3/3 [00:00<00:00, 62.50it/s]
```

```
x_set = df["train"].shuffle(seed=42).select([i for i in list(range(250))])
y_set = df["test"].shuffle(seed=42).select([i for i in list(range(250))])
```

```
print(f'The total number of instances in the train set are {len(x_set)}')
print(f'The total number of instances in the test set are {len(y_set)}')
```

```
The total number of instances in the train set are 250
The total number of instances in the test set are 250
```

The scope of this assignment is to understand, test and analyze the performance of distilBERT transformer model. I'll be using a dataset of 500 rows and dividing them equally. It'll be interesting to see how the results differ in having equal amount of points for each label.

```
#printing the first sentence from the train dataframe
x_set[0]
```

```
{'text': 'There is no relation at all between Fortier and Profiler but the fact that both are police series about violent crimes. Profiler looks crispy, Fortier looks classic. Profiler plots are quite simple. Fortier\'s plot are far more complicated... Fortier looks more like Prime Suspect, if we have to spot similarities... The main character is weak and weirdo, but have "clairvoyance". People like to compare, to judge, to evaluate. How about just enjoying? Funny thing too, people writing Fortier looks American but, on the other hand, arguing they prefer American series (!!!). Maybe it\'s the language, or the spirit, but I think this series is more English than American. By the way, the actors are really good and funny. The acting is not superficial at all...',
 'label': 1}
```

From the above chunk, we can see the first instance of the example in the train set. The label refers to the sentiment of the sentence being positive.

```
y_set[2]
```

```
{'text': 'This movie was so frustrating. Everything seemed energetic and I was totally prepared to have a good time. I at least thought I\'d be able to stand it. But, I was wrong. First, the weird looping? It was like watching "America\'s Funniest Home Videos". The damn parents. I hated them so much. The stereo-typical Latino family? I need to speak with the person responsible for this. We need to have a talk. That little girl who was always hanging on someone? I just hated her and had to mention it. Now, the final scene transcends, I must say. It\'s so gloriously bad and full of badness that it is a movie of its own. What crappy dancing. Horrible and beautiful at once.',
 'label': 0}
```

From the above chunk, we can see the second instance of the test test. The labels refers to the sentiment of the paragraph being 0 which is negative. From an intial read from the human eye, it can be concluded that the sentiment of the paragraph is infact negative.

Preprocessing data points

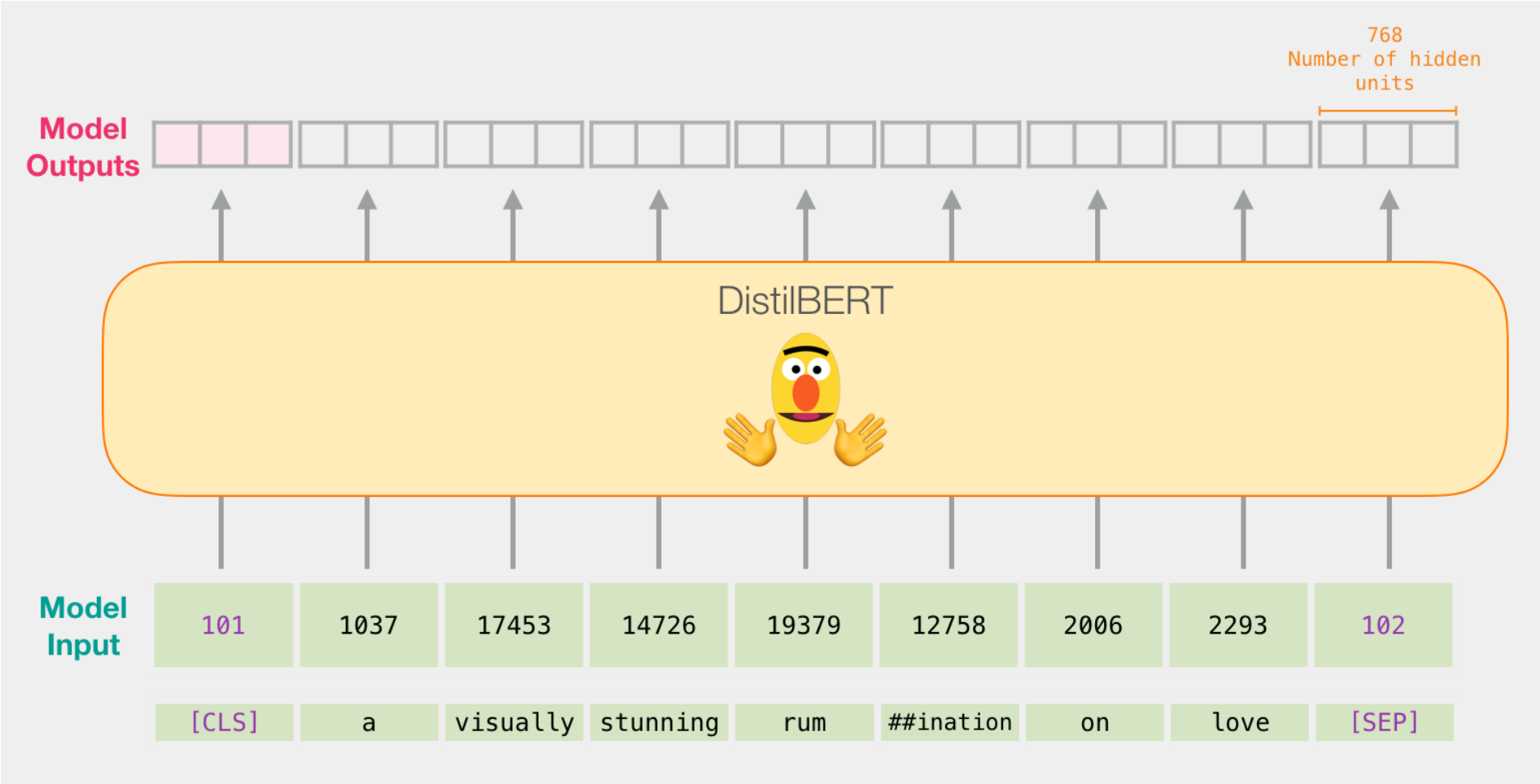
While working with in NLP, the following steps are required to be followed:

1. Tokenize the sentences in the corpus. This is important for translating our sentences into machine friendly language. The sentences are turned into arrays.
2. Padding the sentences in the corpus. Because, there are different lengths of sentences/paragraph present in the dataset, it's vital for all the input paragraphs to be of the same length. This is crucial for model training.

```
#preprocessing our data
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained('distilbert-base-uncased')
```

```
Downloading: 100%  28.0/28.0 [00:00<00:00, 889B/s]
Downloading: 100%  483/483 [00:00<00:00, 10.0kB/s]
Downloading: 100%  232k/232k [00:00<00:00, 1.70MB/s]
Downloading: 100%  466k/466k [00:00<00:00, 1.47MB/s]
```

The preprocessing for NLP tasks working with transformers model requires creating inputs in the following manner:



The input has to be divided into [CLS] which is always 101: the sentence:[SEP]

```
#Prepare the text
def preprocess_text(corpus):
  return tokenizer(corpus["text"], truncation = True)

x_tokenized = x_set.map(preprocess_text, batched = True)
y_tokenized = y_set.map(preprocess_text, batched = True)
```

0% 0/1 [00:00<?, ?ba/s]

0% 0/1 [00:00<?, ?ba/s]

```
x_tokenized[0]
```

```
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,

1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1
```

```
        1,
        1,
        1,
        1,
        1,
        1,
        1,
        1,
        1,
        1,
        1,
        1,
        1,
        1,
        1,
        1,
        1,
        1,
        1,
        1,
        1,
        1]}
```

The above chunk illustrates the shape of the input tensors which will be fed to the model. It includes the output label, input ID's, the attention masks.

- The input_ids: are the specific indices corresponding to each token value in the sentence.
- The attention_masks: are the indicators of whether a word is present or not

Now since we are dealing with less amount of data values we wouldn't need to be worried about the training time. But to make and understand how we can use distilBERT in an actual project, it would be beneficial to convert the training samples to pytorch tensors and concatenate them together to form a corpus with equal length for all of these sentences.

DataCollator are unique objects that will combine together train or test values to build a concrete batch. In order to feed the train and test set values, its required to pad them.

For padding, the maximum length will be for 512 tokens.

```
#to speed up the training process, using data_collator for padding
from transformers import DataCollatorWithPadding
data_collator = DataCollatorWithPadding(tokenizer = tokenizer)
```

Finally, that we have finished our understanding the data values transformed and preprocessed them as the input to be fed to our transformer model, we will be building and training the BERT model. The way transfer learning works is to use the lower half of the model however you want. For training the model, I will be using the trainer API for fine tuning.

The documentation defines the trainer API as a unique API build and made open source by the Hugginface, to fine- tune and train various transformer models such as distilBERT, roBERTa and etc. It's inspired by the PyTorch's ability to train and evaluate models robustly.

The model which I will be experimenting with is the distilbert, a lower heavy version of the actual BERT. I will be using the uncased as it makes all the sentences lowered as the input.

```
#training the model
modelname = 'distilbert-base-uncased'
labels = 2
from transformers import AutoModelForSequenceClassification
model = AutoModelForSequenceClassification.from_pretrained(modelname, num_labels = labels)
```

```
Downloading: 100%                                    268M/268M [00:04<00:00, 57.3MB/s]
Some weights of the model checkpoint at distilbert-base-uncased were not used when initializing DistilB
- This IS expected if you are initializing DistilBertForSequenceClassification from the checkpoint of a
- This IS NOT expected if you are initializing DistilBertForSequenceClassification from the checkpoint
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at d
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inf
```

```
#printing the architecture of the model
model
```

```
(5): TransformerBlock(
  (attention): MultiHeadSelfAttention(
    (dropout): Dropout(p=0.1, inplace=False)
    (q_lin): Linear(in_features=768, out_features=768, bias=True)
```

```
      (k_lin): Linear(in_features=768, out_features=768, bias=True)
      (v_lin): Linear(in_features=768, out_features=768, bias=True)
      (out_lin): Linear(in_features=768, out_features=768, bias=True)
    )
    (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (ffn): FFN(
      (dropout): Dropout(p=0.1, inplace=False)
      (lin1): Linear(in_features=768, out_features=3072, bias=True)
      (lin2): Linear(in_features=3072, out_features=768, bias=True)
      (activation): GELUActivation()
    )
    (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  )
  (4): TransformerBlock(
    (attention): MultiHeadSelfAttention(
      (dropout): Dropout(p=0.1, inplace=False)
      (q_lin): Linear(in_features=768, out_features=768, bias=True)
      (k_lin): Linear(in_features=768, out_features=768, bias=True)
      (v_lin): Linear(in_features=768, out_features=768, bias=True)
      (out_lin): Linear(in_features=768, out_features=768, bias=True)
    )
    (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (ffn): FFN(
      (dropout): Dropout(p=0.1, inplace=False)
      (lin1): Linear(in_features=768, out_features=3072, bias=True)
      (lin2): Linear(in_features=3072, out_features=768, bias=True)
      (activation): GELUActivation()
    )
    (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  )
  (5): TransformerBlock(
    (attention): MultiHeadSelfAttention(
      (dropout): Dropout(p=0.1, inplace=False)
      (q_lin): Linear(in_features=768, out_features=768, bias=True)
      (k_lin): Linear(in_features=768, out_features=768, bias=True)
      (v_lin): Linear(in_features=768, out_features=768, bias=True)
      (out_lin): Linear(in_features=768, out_features=768, bias=True)
    )
    (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (ffn): FFN(
      (dropout): Dropout(p=0.1, inplace=False)
      (lin1): Linear(in_features=768, out_features=3072, bias=True)
```

```
                (lin2): Linear(in_features=3072, out_features=768, bias=True)
                (activation): GELUActivation()
              )
              (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            )
          )
        )
      )
      (pre_classifier): Linear(in_features=768, out_features=768, bias=True)
      (classifier): Linear(in_features=768, out_features=2, bias=True)
      (dropout): Dropout(p=0.2, inplace=False)
    )
```

The last thing to ensure before training the model using trainer API, is to define the evaluation metrics for the model.

```
from datasets import load_metric

def evalute_model(eval_pred):
   accuracy = load_metric("accuracy")
   f1_score = load_metric("f1")

   logits, labels = eval_pred
   predictions = np.argmax(logits, axis=-1)
   accuracy = accuracy.compute(predictions=predictions, references=labels)["accuracy"]
   f1 = f1_score.compute(predictions=predictions, references=labels)["f1"]
   return {"accuracy": accuracy, "f1": f1}
```

In order, to save our trained model, I will be using the my huggingface's notebook credients as described in the tutorial. This was a fairly new thing for me to learn as I wasn't aware that personally trained models can be hosted on huggingface and then later be used for making inference.

```
from huggingface_hub import notebook_login
notebook_login()
```

```
    Login successful
    Your token has been saved to /root/.huggingface/token
```

In order to submit the model on huggingface, it needs to be a part of an output_directory i.e., the directory in which the sentiment analysis takes place. As mentioned in the tutorial, the author mentions about the fine-tuning-sentiment-model. When looked on this repository, I found out that it consists of the different sentiment analysis models that have been trained by different people.

```
from transformers import TrainingArguments, Trainer

#defining the output directory i.e., where the model will be saved for the final inference
repo_name = "finetuning-sentiment-model-3000-samples"
lr = 2e-5
epochs = 2

# The trainer API includes the trainerarguments() function, that takes in the necessary metrics
# for training or fine-tuning the model.
training_args = TrainingArguments(
    output_dir = repo_name,    #the output directory where the model will be submitted
   learning_rate=lr,           # the learning rate with which the model will be trained
   per_device_train_batch_size=16, #the batch size for training
   per_device_eval_batch_size=16,  #the batch size for testing
   num_train_epochs=epochs,        #the number of epochs the model will be trained for
   save_strategy="epoch",          #saving the model's performance after every epoch
   push_to_hub=True,              #for submitting to huggingface
)

# The Trainer() is the function that trained the model while taking the train and test set
# values after the preprocessing and calculates their accuracy and f1score.

trainer = Trainer(
   model=model,          # the model to train
   args=training_args,    #the metrics to fine-tune for the model to perform optimally
   train_dataset=x_tokenized, #the training set
   eval_dataset=y_tokenized,  #the test set
   tokenizer=tokenizer,       # the tokenizer used for conduct preprocessing of data values in the corpus
   data_collator=data_collator, #to build unique batches from the train or test set
   compute_metrics=evalute_model, #the evaluation metrics to be calculated of the model
)
```

```
Cloning https://huggingface.co/mlstudent/finetuning-sentiment-model-3000-samples into local empty direc
WARNING:huggingface_hub.repository:Cloning https://huggingface.co/mlstudent/finetuning-sentiment-model-
```

Download file pytorch_model.bin: 100% 255M/255M [01:34<00:00, 440kB/s]

Download file runs/Oct18_13-30-08_0b5207d5281e/1666099815.125156/events.out.tfevents.1666099815.0b5207d5281e.68.5: 100% 5.40k/5.40k [01:34<00:00, 20.7B/s]

Download file runs/Oct18_13-25-06_0b5207d5281e/events.out.tfevents.1666099513.0b5207d5281e.68.2: 100% 3.99k/3.99k [01:34<00:00, 1.49kB/s]

Download file runs/Oct18_13-15-40_0b5207d5281e/events.out.tfevents.1666098957.0b5207d5281e.68.0: 100% 3.90k/3.90k [01:34<00:00, 1.26kB/s]

Download file runs/Oct18_13-15-40_0b5207d5281e/1666098957.382812/events.out.tfevents.1666098957.0b5207d5281e.68.1: 100% 5.40k/5.40k [01:34<00:00, 9.61kB/s]

Download file training_args.bin: 100% 3.36k/3.36k [01:34<?, ?B/s]

Download file runs/Oct18_13-30-08_0b5207d5281e/events.out.tfevents.1666099815.0b5207d5281e.68.4: 100% 3.99k/3.99k [01:34<00:00, 3.62kB/s]

Download file runs/Oct18_13-25-06_0b5207d5281e/1666099513.0776649/events.out.tfevents.1666099513.0b5207d5281e.68.3: 100% 5.40k/5.40k [01:34<00:00, 12.4kB/s]

```
#Training the model
trainer.train()
```

```
The following columns in the training set don't have a corresponding argument in `DistilBertForSequence
/usr/local/lib/python3.7/dist-packages/transformers/optimization.py:310: FutureWarning: This implementa
  FutureWarning,
***** Running training *****
  Num examples = 250
  Num Epochs = 2
  Instantaneous batch size per device = 16
  Total train batch size (w. parallel, distributed & accumulation) = 16
  Gradient Accumulation steps = 1
  Total optimization steps = 32
You're using a DistilBertTokenizerFast tokenizer. Please note that with a fast tokenizer, using the `__
```

```
[32/32 00:43, Epoch 2/2]
```

Step	Training Loss

```
Saving model checkpoint to finetuning-sentiment-model-3000-samples/checkpoint-16
Configuration saved in finetuning-sentiment-model-3000-samples/checkpoint-16/config.json
Model weights saved in finetuning-sentiment-model-3000-samples/checkpoint-16/pytorch_model.bin
tokenizer config file saved in finetuning-sentiment-model-3000-samples/checkpoint-16/tokenizer_config.j
Special tokens file saved in finetuning-sentiment-model-3000-samples/checkpoint-16/special_tokens_map.j
tokenizer config file saved in finetuning-sentiment-model-3000-samples/tokenizer_config.json
Special tokens file saved in finetuning-sentiment-model-3000-samples/special_tokens_map.json
Saving model checkpoint to finetuning-sentiment-model-3000-samples/checkpoint-32
Configuration saved in finetuning-sentiment-model-3000-samples/checkpoint-32/config.json
Model weights saved in finetuning-sentiment-model-3000-samples/checkpoint-32/pytorch_model.bin
tokenizer config file saved in finetuning-sentiment-model-3000-samples/checkpoint-32/tokenizer_config.j
Special tokens file saved in finetuning-sentiment-model-3000-samples/checkpoint-32/special_tokens_map.j


Training completed. Do not forget to share your model on huggingface.co/models =)


TrainOutput(global_step=32, training_loss=0.6668218374252319, metrics={'train_runtime': 46.5747,
```

```
# Evaluating the model
trainer.evaluate()
```

```
The following columns in the evaluation set don't have a corresponding argument in `DistilBertForSequer
***** Running Evaluation *****
  Num examples = 250
  Batch size = 16
```

```
[16/16 00:03]
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: FutureWarning: load_metric is deprecate
  after removing the cwd from sys.path.
```

Downloading builder script: 4.21k/? [00:00<00:00, 75.4kB/s]

Downloading builder script: 6.50k/? [00:00<00:00, 190kB/s]

```
{'eval_loss': 0.664635181427002,
 'eval_accuracy': 0.632,
 'eval_f1': 0.43209876543209874,
```

From the above chunck, we can see the model's accuracy of 60% and and f1-score of 40% just after training the model for 2 epochs. With only 250 samples present for each train and test sets.

Now, that we have understood the architecture, and how to use Trainer API to preprocess, fine-tune, train and evaluate the model. By using trainer.push_to_hub(), the model will be submitted on huggingface.

```
trainer.push_to_hub()
```

```
Saving model checkpoint to finetuning-sentiment-model-3000-samples
Configuration saved in finetuning-sentiment-model-3000-samples/config.json
Model weights saved in finetuning-sentiment-model-3000-samples/pytorch_model.bin
tokenizer config file saved in finetuning-sentiment-model-3000-samples/tokenizer_config.json
Special tokens file saved in finetuning-sentiment-model-3000-samples/special_tokens_map.json
Several commits (2) will be pushed upstream.
WARNING:huggingface_hub.repository:Several commits (2) will be pushed upstream.
The progress bars may be unreliable.
WARNING:huggingface_hub.repository:The progress bars may be unreliable.
```

Upload file pytorch_model.bin: 100% 255M/255M [06:26<00:00, 2.13MB/s]

Upload file runs/Oct18_19-42-33_5f067d55e9b9/events.out.tfevents.1666122310.5f067d55e9b9.71.2: 100% 402/402 [06:26<?, ?B/s]

Upload file runs/Oct18_19-42- 3.90k/3.90k

Lastly, to use the model trained above for inference using pipeline from transformer library. I will be using adding my username/the output directory used in the earlier stages of the model.

```
remote: LFS file scan complete
```

```
movie_sentiment = pipeline(model = 'mlstudent/finetuning-sentiment-model-3000-samples')
movie_sentiment('Today I have completed my NLP assignment. Understood the implementation of distilBERT using Trainer API')
```

```
Downloading: 100%                                                    615/615 [00:00<00:00, 4.99kB/s]
loading configuration file config.json from cache at /root/.cache/huggingface/hub/models--mlstudent--fi
Model config DistilBertConfig {
  "_name_or_path": "mlstudent/finetuning-sentiment-model-3000-samples",
  "activation": "gelu",
  "architectures": [
    "DistilBertForSequenceClassification"
  ],
  "attention_dropout": 0.1,
  "dim": 768,
  "dropout": 0.1,
  "hidden_dim": 3072,
  "initializer_range": 0.02,
  "max_position_embeddings": 512,
  "model_type": "distilbert",
  "n_heads": 12,
  "n_layers": 6,
  "pad_token_id": 0,
  "problem_type": "single_label_classification",
  "qa_dropout": 0.1,
  "seq_classif_dropout": 0.2,
  "sinusoidal_pos_embds": false,
  "tie_weights_": true,
  "torch_dtype": "float32",
  "transformers_version": "4.23.1",
  "vocab_size": 30522
}

Downloading: 100%                                                    3.00k/3.00k [00:00<00:00, 36.7kB/s]
loading configuration file config.json from cache at /root/.cache/huggingface/hub/models--mlstudent--fi
Model config DistilBertConfig {
  "_name_or_path": "mlstudent/finetuning-sentiment-model-3000-samples",
  "activation": "gelu",
  "architectures": [
    "DistilBertForSequenceClassification"
  ],
  "attention_dropout": 0.1,
  "dim": 768,
  "dropout": 0.1,
  "hidden_dim": 3072,
  "initializer_range": 0.02,
```

```
  "initializer_range": 0.02,
  "max_position_embeddings": 512,
  "model_type": "distilbert",
  "n_heads": 12,
  "n_layers": 6,
  "pad_token_id": 0,
  "problem_type": "single_label_classification",
  "qa_dropout": 0.1,
  "seq_classif_dropout": 0.2,
  "sinusoidal_pos_embds": false,
  "tie_weights_": true,
  "torch_dtype": "float32",
  "transformers_version": "4.23.1",
  "vocab_size": 30522
}
```


Downloading: 100% 268M/268M [00:04<00:00, 51.0MB/s]

```
loading weights file pytorch_model.bin from cache at /root/.cache/huggingface/hub/models--mlstudent--fi
All model checkpoint weights were used when initializing DistilBertForSequenceClassification.
```

The model predicts the above sentence as a `label` 0 which is negative.

Downloading: 100% 360/360 [00:00<00:00, 10.8kB/s]

```
movie_sentiment('The weekend is amazing!')
```

```
[{'label': 'LABEL_1', 'score': 0.5169048309326172}]
```

The model predicted the above sentence as `label 1` which is positive. This is a good result which is probably because of the exclamation point.

```
movie_sentiment('The yoghurt is ridiculosly good.')
```

```
[{'label': 'LABEL_1', 'score': 0.5030409097671509}]
```

The model predicted the above sentence as `label 1` which is positive. This means the model did a fair job to understand sentiment with about 50% accuracy.

Conclusion & Recommendation

By following the tutorial from the huggingface library, I have definetely learned how to implement pyTorch tensors in training or fine-tuning a BERT model. Additionally, learning about the use of Trainer API is something, I'll be using in my future projects because its easy to understand and implement. The documentation of the huggingface is done in a good manner and easy to follow.

For future recommendations, I would like to try the model on larger dataset and see how the metrics change.