Fine-Tuning Bert For Classification

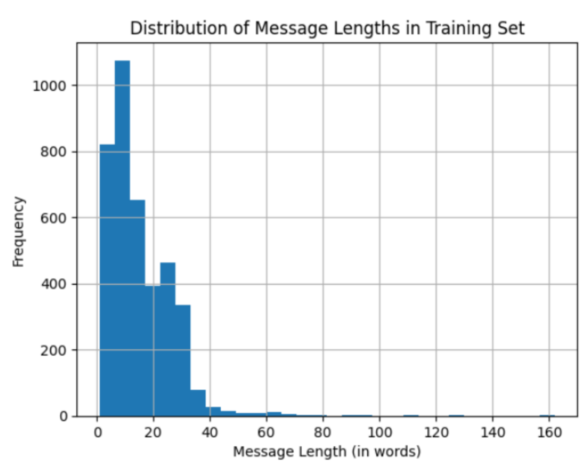
Project Overview

In this project, we are going to use the pre-trained open-source BERT model and fine-tune it to perform our required task. In this case, it is a classification problem.

Data-Visualisation, Pre-Processing & Feature Engineering

label	text
0	You will be in the place of that man
1	Bears Pic Nick, and Tom, Pete and ... Dick. In...
0	Lol they were mad at first but then they woke ...
0	Good morning princess! How are you?
0	Heart is empty without love.. Mind is empty wi...

1. We observe that each row contains a text and a label indicating '0' or '1'. Thus, this is a `BINARY CLASSIFICATION PROBLEM`.
2. As part of basic preprocessing, we drop any rows that are duplicates or contain null values.
3. Always check for **class imbalance**, an important consideration in classification tasks. To address class imbalance, we assign a higher penalty when the model misclassifies a minority class during training. This approach encourages the model to pay attention to minority classes, even if the overall accuracy is high.
4. Split the data into train, validation, and test sets in a 70:15:15 ratio. We will fine-tune the model using the train set and the validation set, and make predictions for the test set.
5. Import the BERT-base pre-trained model and load the BERT tokenizer, both of which are uncased.
6. Now, as the input length of the BERT model is limited to a maximum of 512 tokens, and all inputs in a given batch have to be of the same length, choosing an appropriate `max_length` based on the typical length distribution of your dataset helps retain the most relevant information.



```
Minimum words in a sentence: 1
Maximum words in a sentence: 162
Most common sentence length (words): 6 with frequency: 308
Least common sentence length (words): 66 with frequency: 1
```

7. We try to pad smaller sentences and truncate the longer ones. Hence, I choose an optimal `max_length` of 25, which I will use in batches, so I applied `batch_encode`.
8. We will be using the PyTorch framework, so we convert these input IDs, attention masks, and labels to tensors.
9. To send training data in batches and shuffle it every epoch, we use `DataLoader` and samplers.
10. Although there are many ways to fine-tune a large language model (LLM), we are going to use the third approach mentioned below:
    (a) Train all weights.
    (b) Freeze a few layers and train the remaining unfreezed layer weights.
    (c) Freeze all layers, add new layers, and train only those newly added layers.

Model-Architecture

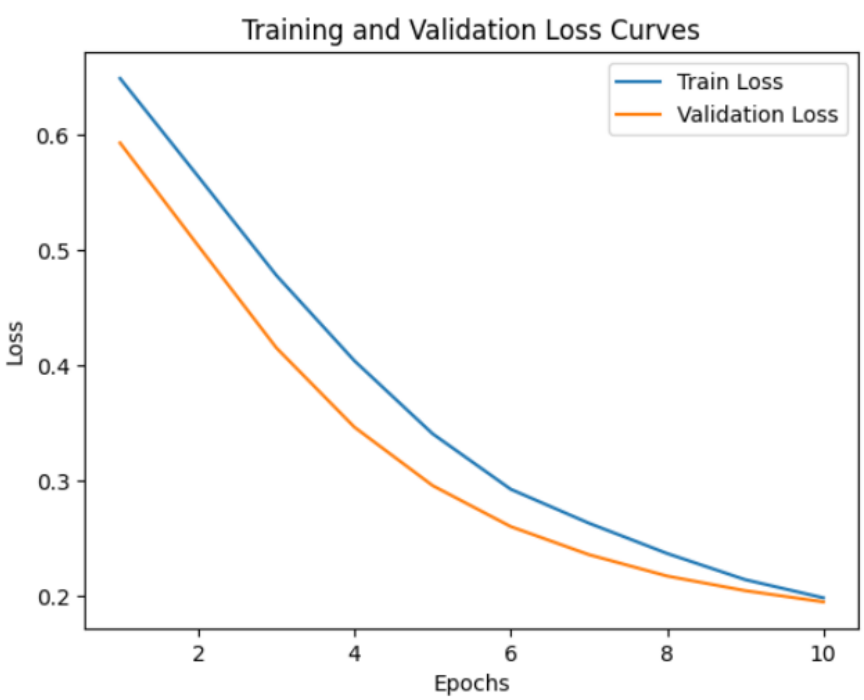
Input → BERT → Dense(768→512) → ReLU → Dropout → Dense(512→2) → LogSoftmax → Output

```
BERT_Arch(
(bert): 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(
      (0-11): 12 x BertLayer(
        (attention): BertAttention(
          (self): BertSdpaSelfAttention(
            (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()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (relu): ReLU()
  (fc1): Linear(in_features=768, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=2, bias=True)
  (softmax): LogSoftmax(dim=1)
)
```

Model-Training

1. Optimizer used id AdamW with learning rate set to 1e-5
2. Could use a schedular as well, but the data is very less so ignored.
3. loss function as cross_entropy (taking care of class imbalance)
4. No of training epochs = 10



Model-Evaluation

	precision	recall	f1-score	support
0	0.98	0.97	0.98	724
1	0.81	0.90	0.86	112
accuracy			0.96	836
macro avg	0.90	0.94	0.92	836
weighted avg	0.96	0.96	0.96	836