NLP Engineer Assignment report submission

→ A brief introduction to the task and the dataset used

The goal of this assignment is to build a text classification model using the Hugging Face library to classify a dataset of text into one of multiple categories. By using a pre-trained model such as BERT or GPT-2 as a starting point and fine-tune it on the classification task.

- Fine-tune A NLP Model: Fine-tuning a natural language processing (NLP) model involves adjusting the hyperparameters and architecture of the model, and often also involves adjusting the dataset, to improve the performance of the model on a specific task

- About Hugging Face : Hugging Face is a company that provides a platform for training and deploying natural language processing (NLP) models. The platform includes a library of pre-trained models that can be used for a variety of NLP tasks, such as language translation, text generation, and question answering.

Dataset contains:

- 45500 rows and 5 columns
- Target column: Category (Business , Politics, Food & Drink, TRAVEL ,Parenting, STYLE & BEAUTY ,Wellness, World news, Sports , Entertainment)

  -Each category class contains 4500 rows

  -It contains nan values only in keywords column
- Apart from that, the original dataset had lots of third person statements (like "This statement is irrelevant" says the officials)

  -Keyword column has been added where main keywords in a url are extracted (urls were in the original dataset)

→ The preprocessing steps taken :

- Preprocessed the data by removing stopwords, punctuations and lemmatized using WordNetLemmatizer

→ The architecture of the model used, and how it was fine-tuned

```
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to                     
==================================================================================================
 input_ids (InputLayer)         [(None, 8)]          0           []                               
                                                                                                  
 attention_mask (InputLayer)    [(None, 8)]          0           []                               
                                                                                                  
 tf_distil_bert_model (TFDistil  TFBaseModelOutput(l  66362880   ['input_ids[0][0]',              
 BertModel)                     ast_hidden_state=(N               'attention_mask[0][0]']         
                                one, 8, 768),                                                     
                                 hidden_states=None                                               
                                , attentions=None)                                                
                                                                                                  
 global_max_pooling1d (GlobalMa  (None, 768)         0           ['tf_distil_bert_model[0][0]']   
 xPooling1D)                                                                                      
                                                                                                  
 dense (Dense)                  (None, 128)          98432       ['global_max_pooling1d[0][0]']   
                                                                                                  
 dropout_19 (Dropout)           (None, 128)          0           ['dense[0][0]']                  
                                                                                                  
 dense_1 (Dense)                (None, 32)           4128        ['dropout_19[0][0]']             
                                                                                                  
 dense_2 (Dense)                (None, 10)           330         ['dense_1[0][0]']                
                                                                                                  
==================================================================================================
Total params: 66,465,770
Trainable params: 66,465,770
Non-trainable params: 0
```

```
Epoch 1/5
250/250 [==============================] - 138s 549ms/step - loss: 1.4249 - balanced_accuracy: 0.5410 - val_loss: 1.1163 - val_
balanced_accuracy: 0.6650
Epoch 2/5
250/250 [==============================] - 142s 570ms/step - loss: 0.8850 - balanced_accuracy: 0.7305 - val_loss: 1.0718 - val_
balanced_accuracy: 0.6900
Epoch 3/5
250/250 [==============================] - 147s 590ms/step - loss: 0.7220 - balanced_accuracy: 0.7809 - val_loss: 1.0863 - val_
balanced_accuracy: 0.7000
Epoch 4/5
250/250 [==============================] - 133s 533ms/step - loss: 0.6216 - balanced_accuracy: 0.8190 - val_loss: 1.1012 - val_
balanced_accuracy: 0.6933
Epoch 5/5
250/250 [==============================] - 123s 490ms/step - loss: 0.5423 - balanced_accuracy: 0.8384 - val_loss: 1.1197 - val_
balanced_accuracy: 0.7017
```

→ A discussion of the performance of the model and possible ways to improve it

- As epochs increased model performance also increased. And we got very less loss in both train and test
- By using a higher **Batch Size,** We can increase the model accuracy. I have used batch_size=8. Due to my system GPU is running out of memory.

→ Sample predictions and their explanations

```
# Test case - 3

texts = str("alanis morissette ditch tough image softer look photo look never go style")
test_tokenized = tokenizer(text=texts, padding="max_length", truncation=True, max_length=8, return_tensors = "tf")
validate = model.predict({'input_ids' :test_tokenized['input_ids'], 'attention_mask' :test_tokenized['attention_mask']})*100
for key,value in zip(dict_val.keys(), validate[0]):
    print(key,value)
```

```
1/1 [==============================] - 0s 156ms/step
BUSINESS 0.2898894
ENTERTAINMENT 2.7501802
FOOD & DRINK 0.32724157
PARENTING 0.12363454
POLITICS 0.059796054
SPORTS 0.05484691
STYLE & BEAUTY 95.56508
TRAVEL 0.16658288
WELLNESS 0.44026262
WORLD NEWS 0.22248751
```

And you can see more in the code.

DATASET LINK : https://www.kaggle.com/datasets/setseries/news-category-dataset