Building an AI Powered spam Classifier

Phase 4: Development part 2

Introduction:

Developing an effective Spam SMS Detection model involves training a machine learning algorithm to automatically identify and filter out unwanted, unsolicited text messages from legitimate ones. Also it includes training the model, and evaluating its performance.

Dataset Used:

https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset

Building the Model and Predicting the Data

BERT:

Encoder-Only Models:

Function: Encoder-only models are designed to process input sequences and encode them into a fixed-length representation, often referred to as a context vector or latent representation. They do not have a decoding mechanism.

Usage Example: BERT (Bidirectional Encoder Representations from Transformers) is a popular encoder-only model. It's pre-trained on a massive corpus of text and used for various downstream NLP tasks like text classification, named entity recognition, and question answering. Decoder-Only Models.

```
import tensorflow as tf
from tensorflow import keras
from keras.layers import Dense  , Input
from keras.optimizers import Adam
from keras.models import Model
from keras.callbacks import ModelCheckpoint

import transformers
from transformers import BertTokenizer , TFBertModel
```

```
tokenizer = BertTokenizer.from_pretrained('bert-large-uncased')
bert_model= TFBertModel.from_pretrained('bert-large-uncased')
```

let's create a function to encode our input text to extract 'input_ids', and 'attention_maske'

```
def encode_(data , max_len):
    ids_list = []
    mask_list= []

    for text in data:
        text_encode = tokenizer.encode_plus(

            text ,
            max_length = max_len ,
            add_special_tokens=True ,
            truncation=True,
            padding='max_length' ,
            return_attention_mask=True
        )
        ids_list.append(text_encode['input_ids'])
        mask_list.append(text_encode['attention_mask'])
    return np.array(ids_list) , np.array(mask_list)
```

```
train_input_ids , train_attention_mask = encode_(train_data ,60)
```

Creating the BERT Model

```
def create_bert_model(bert_model):
    input_ids     = Input(shape=(60,) , dtype='int32')
    attention_mask= Input(shape=(60,) , dtype='int32')
    ### layers
    out = bert_model(input_ids ,attention_mask)[1]
    out = Dense(32 , activation='relu')(out)
    out = Dropout(0.2)(out)
    out = Dense(1 , activation ='sigmoid')(out)
    ### initiate the model
    model = Model(inputs = [input_ids , attention_mask] , outputs = out)

    model.compile(Adam(learning_rate=1e-5) , loss =  'binary_crossentropy'
, metrics = ['accuracy'])

    return model
```

```
Bert_model = create_bert_model(bert_model)
Bert_model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 60)]         0           []

 input_2 (InputLayer)           [(None, 60)]         0           []

 tf_bert_model (TFBertModel)    TFBaseModelOutputWi  335141888   ['input_1[0][0]',
                                thPoolingAndCrossAt               'input_2[0][0]']
                                tentions(last_hidde
                                n_state=(None, 60,
                                1024),
                                 pooler_output=(Non
                                e, 1024),
                                 past_key_values=No
                                ne, hidden_states=N
                                one, attentions=Non
                                e, cross_attentions
                                =None)

 dense (Dense)                  (None, 32)           32800       ['tf_bert_model[0][1]']

 dropout_73 (Dropout)           (None, 32)           0           ['dense[0][0]']

 dense_1 (Dense)                (None, 1)            33          ['dropout_73[0][0]']

==================================================================================================
Total params: 335,174,721
Trainable params: 335,174,721
Non-trainable params: 0
__________________________________________________________________________________________________
```

Training the Model

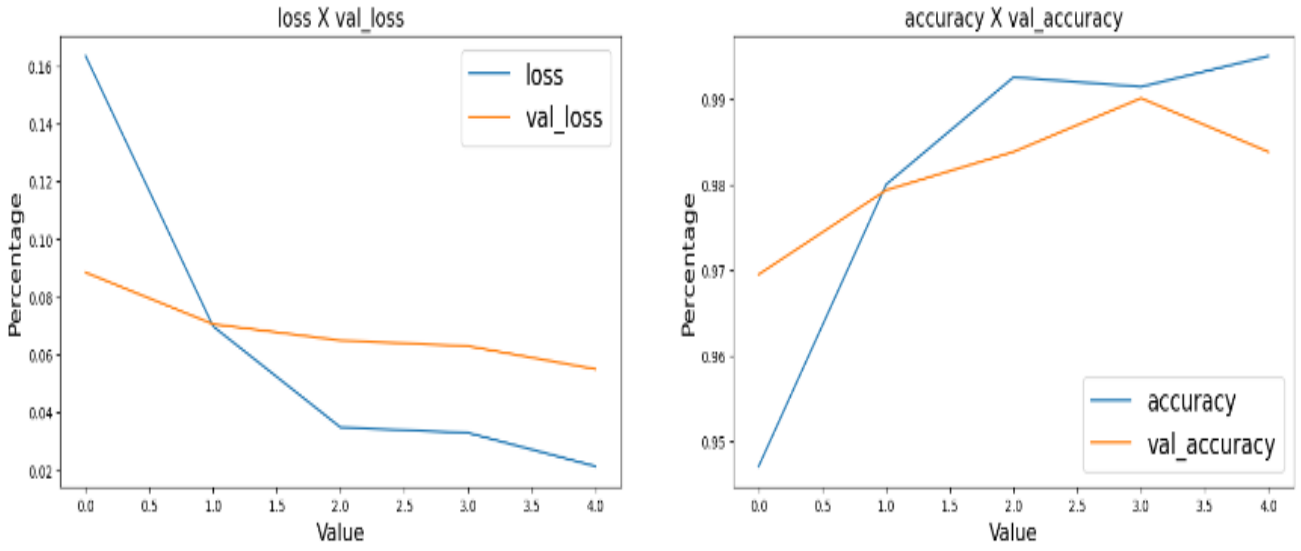
```
history = Bert_model.fit(
    [train_input_ids , train_attention_mask] ,
    y_label ,
    validation_split = 0.2,
    epochs = 5 ,
    batch_size=16

)
```

```
Epoch 1/5
279/279 [==============================] - 161s 468ms/step - loss: 0.1634 - accuracy: 0.9470 - val
_loss: 0.0884 - val_accuracy: 0.9695
Epoch 2/5
279/279 [==============================] - 135s 485ms/step - loss: 0.0698 - accuracy: 0.9800 - val
_loss: 0.0706 - val_accuracy: 0.9794
Epoch 3/5
279/279 [==============================] - 135s 485ms/step - loss: 0.0348 - accuracy: 0.9926 - val
_loss: 0.0649 - val_accuracy: 0.9839
Epoch 4/5
279/279 [==============================] - 135s 485ms/step - loss: 0.0329 - accuracy: 0.9915 - val
_loss: 0.0629 - val_accuracy: 0.9901
Epoch 5/5
279/279 [==============================] - 135s 485ms/step - loss: 0.0213 - accuracy: 0.9951 - val
_loss: 0.0550 - val_accuracy: 0.9839
```

```
def plot_history(history, arr):
    fig, ax = plt.subplots(1, 2, figsize=(20, 5))
    for index in range(2):
        ax[index].plot(history.history[arr[index][0]])
        ax[index].plot(history.history[arr[index][1]])
        ax[index].legend([arr[index][0], arr[index][1]],fontsize=18)
        ax[index].set_xlabel('Value',fontsize=16)
        ax[index].set_ylabel('Percentage',fontsize=16)
        ax[index].set_title(arr[index][0] + ' X ' + arr[index][1],fontsize=16)
```

```
plot_history(history , [['loss', 'val_loss'],['accuracy', 'val_accuracy'] ])
```



Test the Model with Zero shot inference

```
preamble = "Classify the following SMS message as either 'ham' or 'spam':"
end_prompt   = 'Answer:\n'

template =  Template('$preamble\n\n$prompt\n\n$end_prompt')



def create_prompt(data , index):
    prompt = data.loc[index , 'Message']

    input_text = template.substitute(preamble=preamble , prompt = prompt ,end_prompt=end_prompt)
    return input_text
```

```
print(create_prompt(data , 0))
```

```
print(create_prompt(data , 0))
```

```
Classify the following SMS message as either 'ham' or 'spam':

Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got
amore wat...

Answer:
```

```
import random
index = random.randint(0,200)

zero_shot_message = create_prompt(data , index)
zero_shot_class   = data.loc[index , 'Target']


inputs  = tokenizer(zero_shot_message , return_tensors='pt')
generate= Flan_model.generate(inputs['input_ids'] )[0]
model_answer = tokenizer.decode(generate , skip_special_tokens=True)


line_dash = '-'.join('' for _ in range(100))
print(line_dash)
print(line_dash)
print(f'Prompt:\n{zero_shot_message}')
print(line_dash)
print(f'Acual Answer:\n{zero_shot_class}')
print(line_dash)
print(f'Model Answer:\n{model_answer}')
```

```
----------------------------------------------------------------------------------------------------
-
----------------------------------------------------------------------------------------------------
-
Prompt:
Classify the following SMS message as either 'ham' or 'spam':

FreeMsg Why haven't you replied to my text? I'm Randy, sexy, female and live local. Luv to hear fr
om u. Netcollex Ltd 08700621170150p per msg reply Stop to end
```

```
Answer:

--------------------------------------------------------------------------------------------
-
Acual Answer:
spam
--------------------------------------------------------------------------------------------
-
Model Answer:
spam
```

Conclusion:

Thus the project is build, trained and also it is tested with some sample input.