Arabic dialect Prediction

This notebook aims to build a model that predicts the dialect given the text.First by attempting some classical ML models .Then moving to deep learning approach through finetuning an Multi-dialect-Arabic-BERT trained on 10M arabic tweets.

+ Code + Markdown

Table of Contents:

+ Code + Markdown

[2]:
```
import os
import re
from tqdm import tqdm
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import csv

%matplotlib inline
```

Load dataset

[3]:
```
import pandas as pd
NEW_DF = pd.read_csv('../input/dataset/Data.csv')
```

+ Code + Markdown

Classical ML approach

Exploratory data analysis(EDA)

[4]:
```
NEW_DF.shape
```

[4]:
```
(458197, 2)
```

[5]:
```
NEW_DF.dialect.unique()
```

[5]:
```
array(['IQ', 'LY', 'QA', 'PL', 'SY', 'TN', 'JO', 'MA', 'SA', 'YE', 'DZ',
       'EG', 'LB', 'KW', 'OM', 'SD', 'AE', 'BH'], dtype=object)
```

[6]:
```
dialect_count = NEW_DF.groupby('dialect', as_index=False).count()
dialect_count.sort_values(['text'],ascending=False,)
```

[6]:

	dialect	text
3	EG	57636
11	PL	43742
6	KW	42109
8	LY	36499
12	QA	31069
5	JO	27921
7	LB	27617
13	SA	26832
0	AE	26296
1	BH	26292
10	OM	19116
15	SY	16242
2	DZ	16183
4	IQ	15497
14	SD	14434
9	MA	11539
17	YE	9927
16	TN	9246

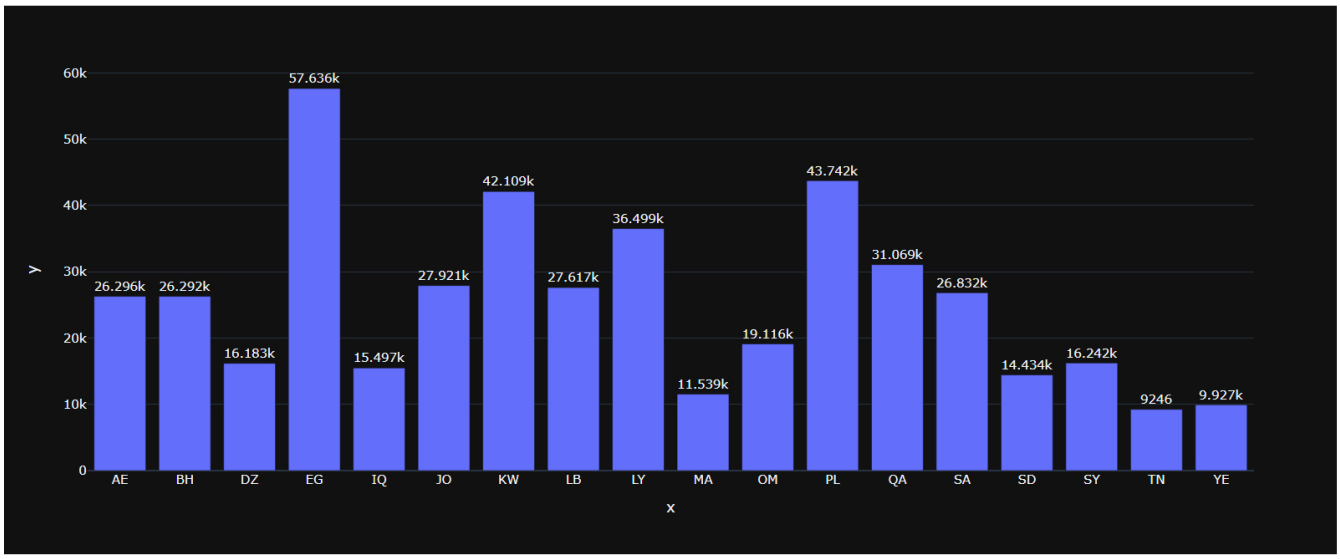
+ Code + Markdown

[7]:

```
import plotly.express as px

Fig1 =px.bar(x=dialect_count.dialect ,y=dialect_count.text, template='plotly_dark')

Fig1.update_traces(  texttemplate="%{y}",textposition='outside')
Fig1.show()
```



+ Code + Markdown

- our data is Imbalanced and that can cause a lot of frustration.
- we can try Resampling our Dataset

[8]:

```
#using over-sampling method that can add more copies to the minority class.

under_balanced_df = pd.DataFrame()
under_balanceed_label =['OM','SY','DZ','IQ','SD','MA','YE','TN']

for lebel in under_balanceed_label:
    under_balanced_df=under_balanced_df.append(NEW_DF[NEW_DF.dialect==lebel].sample(25000, replace=True)).reset_index(drop=True)
under_balanced_df.shape
```

[8]:
```
(200000, 2)
```

+ Code + Markdown

[9]:

```
#using under-sampling method that can delete instances from the over-represented class.
over_balanceed_label=['LY', 'QA', 'PL', 'JO','SA','EG','LB','KW','AE','BH']
over_balanced_df = pd.DataFrame()

for lebel in over_balanceed_label:
    over_balanced_df=over_balanced_df.append(NEW_DF[NEW_DF.dialect==lebel].sample(25000, replace=True)).reset_index(drop=True)
over_balanced_df.shape
```

[9]: (250000, 2)

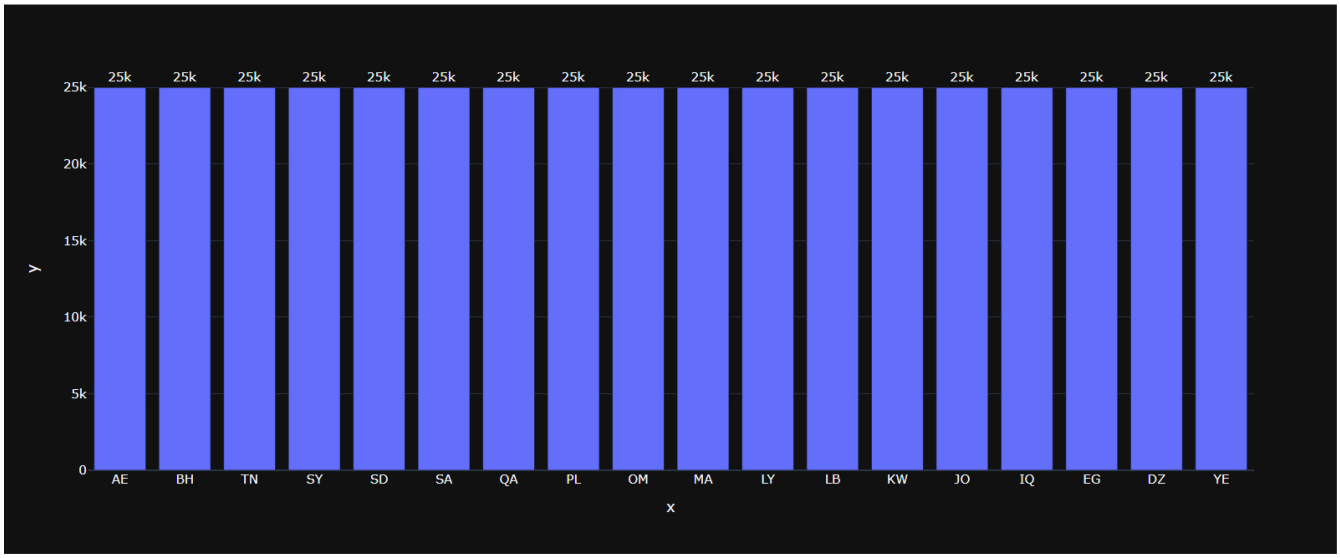
[10]:
```
balanced_df = pd.concat([over_balanced_df, under_balanced_df], axis=0).reset_index(drop=True)
balanced_df.shape
```

[10]: (450000, 2)

+ Code + Markdown

[11]:
```
import plotly.express as px
dialect_count_balanced = balanced_df.groupby('dialect', as_index=False).count().sort_values(['text'],ascending=False,)
Fig1 =px.bar(x=dialect_count_balanced.dialect ,y=dialect_count_balanced.text, template='plotly_dark')

Fig1.update_traces(  texttemplate="%{y}",textposition='outside')
Fig1.show()
```



+ Code + Markdown

Text preprocessing

[12]:
```
import re
import pandas as pd
def text_preprocessing(text):


    """
    @param    text (str): a string to be processed.
    @return   text (Str): the processed string.
    """

    # Remove'@name'
    text = re.sub("@\S+", '', text)
    #Remove URLs
    text = re.sub('^https?:\/\/.*[\r\n]*', '', text)
    # Remove punctuation
    text = re.sub('[!"#$%&\'()*+,-./:;<=>?[\]^_`{|}~]','',text)
    #Remove newline character
    text = re.sub('\n', '', text)
    #Remove emoji
    text = re.sub("[" u"\U0001F600-\U0001F64F"  # emoticons
                            u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                            u"\U0001F680-\U0001F6FF"  # transport & map symbols
                            u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                            u"\U00002702-\U000027B0"
                             u"\U00002500-\U00002BEF"  # chinese char
                                u"\U00002702-\U000027B0"
                                u"\U000024C2-\U0001F251"
                                u"\U0001f926-\U0001f937"
                                u"\U00010000-\U0010ffff"
                                u"\u2640-\u2642"
                                u"\u2600-\u2B55"
                                u"\u200d"
                                u"\u23cf"
                                u"\u23e9"
                                u"\u231a"
                                u"\ufe0f"  # dingbats
                                u"\u3030"
                            u"\U000024C2-\U0001F251"
                            "]+", '', text)
    # replcae (ḷ,ī,ḯ) by (ḷ)
    text = re.sub('[ḷ]', 'ḷ', text)
    # replace (ṡ) by (ṣ)
```

```
    text = re.sub('[ؤ]', 'ه', text)
    text = text.replace('ة' ,'ه')
    text = text.replace('و' ,'وو')
    text = text.replace('ي' ,'يي')
    text = text.replace('ا' ,'اا')
    # remove multi spaces
    text = re.sub(' +', ' ', text)
    return text
```

+ Code + Markdown

[13]:
```
balanced_df['Clean_text']=balanced_df['text'].apply(text_preprocessing)
```

[14]:
```
balanced_df.head()
```

[14]:

	text	dialect	Clean_text
0	@Hisham_Eljamal الحمدلله جت هذي ولا غيرها	LY	الحمدلله جت هذي ولا غيرها
1	@Munir_elmhmoodi هههههههههه لالا ولله مشيت لع...	LY	هههههه لالا ولله مشيت لعبت معاهم
2	" صدقو لك فقالو " كل البنات حلوين https://t.co...	LY	صدقو لك فقالو كل البنات حلوين httpstcohRP7G95PKk
3	@mustafa_agha مش مايل وبس منيل بستين مليون نيل مش...	LY	مش مايل وبس منيل بستين مليون نيله
4	كن يديرو قول اخر تنوض نوضة السنين	LY	كن يديرو قول اخر تنوض نوضه السنين

[15]:
```
from sklearn.model_selection import train_test_split

X = balanced_df.text
y = balanced_df.dialect
x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=42,stratify=y,test_size=0.3)
X_train, X_val, y_train1, y_val = train_test_split(x_train, y_train, random_state=42,stratify=y_train,test_size=0.2)
```

+ Code + Markdown

RandomForest Classifier

[16]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import r2_score
from sklearn.pipeline import Pipeline
from sklearn import metrics
from sklearn.feature_extraction.text import TfidfVectorizer

Regressor = RandomForestClassifier(n_estimators=400,max_depth=35,random_state=128,max_features='sqrt',bootstrap=False)

pipe_reg = Pipeline([
    ('tfid', TfidfVectorizer(ngram_range=(1, 3),)),
    ('model', Regressor)])

pipe_reg.fit(X_train, y_train1)


y_pred_pipe_reg_tr = pipe_reg.predict(X_train)
y_pred_pipe_reg_val = pipe_reg.predict(X_val)
```

+ Code + Markdown

For validation dataset

[17]:
```
target = NEW_DF['dialect'].astype('category')
print('categories: {}'.format(target.cat.categories))
from sklearn import metrics
print(metrics.classification_report(y_val, y_pred_pipe_reg_val))
```

```
categories: Index(['AE', 'BH', 'DZ', 'EG', 'IQ', 'JO', 'KW', 'LB', 'LY', 'MA', 'OM', 'PL',
       'QA', 'SA', 'SD', 'SY', 'TN', 'YE'],
      dtype='object')
              precision    recall  f1-score   support

          AE       0.60      0.40      0.48      3500
          BH       0.81      0.29      0.43      3500
          DZ       0.64      0.63      0.64      3500
          EG       0.48      0.69      0.57      3500
          IQ       0.55      0.65      0.59      3500
          JO       0.63      0.31      0.41      3500
          KW       0.34      0.40      0.37      3500
          LB       0.51      0.68      0.58      3500
          LY       0.59      0.51      0.54      3500
          MA       0.66      0.76      0.70      3500
          OM       0.27      0.55      0.37      3500
          PL       0.52      0.29      0.37      3500
          QA       0.54      0.53      0.53      3500
          SA       0.26      0.42      0.32      3500
          SD       0.65      0.63      0.64      3500
          SY       0.66      0.43      0.52      3500
          TN       0.78      0.62      0.69      3500
          YE       0.49      0.40      0.44      3500
```

```
    accuracy                           0.51     63000
   macro avg       0.55      0.51      0.51     63000
weighted avg       0.55      0.51      0.51     63000
```

+ Code + Markdown

For test dataset

```
y_pred_pipe_reg_test= pipe_reg.predict(x_test)
```

```
y_test
```

[26]:
```
327206    IQ
168075    LB
82869     JO
251354    OM
35641     QA
          ..
282185    SY
107276    SA
104345    SA
53373     PL
429200    TN
Name: dialect, Length: 135000, dtype: object
```

+ Code + Markdown

```
target = NEW_DF['dialect'].astype('category')
print('categories: {}'.format(target.cat.categories))
from sklearn import metrics
print(metrics.classification_report(y_test, y_pred_pipe_reg_test))
```

```
categories: Index(['AE', 'BH', 'DZ', 'EG', 'IQ', 'JO', 'KW', 'LB', 'LY', 'MA', 'OM', 'PL',
       'QA', 'SA', 'SD', 'SY', 'TN', 'YE'],
      dtype='object')
              precision    recall  f1-score   support

          AE       0.60      0.38      0.47      7500
          BH       0.81      0.29      0.43      7500
          DZ       0.63      0.62      0.62      7500
          EG       0.48      0.70      0.57      7500
          IQ       0.55      0.64      0.59      7500
          JO       0.64      0.31      0.42      7500
          KW       0.35      0.41      0.37      7500
          LB       0.49      0.66      0.57      7500
          LY       0.58      0.50      0.54      7500
          MA       0.65      0.75      0.70      7500
          OM       0.27      0.53      0.36      7500
          PL       0.51      0.29      0.37      7500
          QA       0.53      0.53      0.53      7500
          SA       0.26      0.41      0.32      7500
          SD       0.65      0.63      0.64      7500
          SY       0.63      0.41      0.50      7500
          TN       0.78      0.63      0.70      7500
          YE       0.47      0.40      0.43      7500

    accuracy                           0.51    135000
   macro avg       0.55      0.51      0.51    135000
weighted avg       0.55      0.51      0.51    135000
```

+ Code + Markdown

[ ]:

Deep Learning Approach

- Given that the Random Forest Classifier model wasn't generalizing well for other datasets (possibly overfitting), I decided to try a DL approach using a pretrained model (i.e: increasing the dataset as a way of overcoming overfitting). For that I chose to use the Multi-dialect-Arabic-BERT

  > The models were pretrained on 10M arabic tweets

+ Code + Markdown

[11]:
```
import torch

if torch.cuda.is_available():
    device = torch.device("cuda")
    print(f'There are {torch.cuda.device_count()} GPU(s) available.')
    print('Device name:', torch.cuda.get_device_name(0))

else:
    print('No GPU available, using the CPU instead.')
    device = torch.device("cpu")
```

```
There are 1 GPU(s) available.
Device name: Tesla P100-PCIE-16GB
```

[12]:
```
from transformers import AutoTokenizer, AutoModel
```

+ Code + Markdown

Multi-dialect-Arabic-BERT

Code adapted from https://skimai.com/fine-tuning-bert-for-sentiment-analysis/

[13]:
```
tokenizer = AutoTokenizer.from_pretrained("bashar-talafha/multi-dialect-bert-base-arabic")
```

```
Downloading: 100% 456/456 [00:00<00:00, 15.9kB/s]
Downloading: 100% 334k/334k [00:00<00:00, 606kB/s]
```

Preprocessing

[14]:
```
# Define preprocessing util function
def text_preprocessing(text):
    """
    - Remove entity mentions (eg. '@united')
    - Correct errors (eg. '&amp;' to '&')
    @param    text (str): a string to be processed.
    @return   text (Str): the processed string.
    """


    # Normalize unicode encoding
    text = unicodedata.normalize('NFC', text)
    # Remove '@name'
    text = re.sub(r'(@.*?)[\s]', ' ', text)

    # Replace '&amp;' with '&'
    text = re.sub(r'&amp;', '&', text)

    # Remove trailing whitespace
    text = re.sub(r'\s+', ' ', text).strip()

    #Remove URLs
    text = re.sub(r'^https?:\/\/.*[\r\n]*', '<URL>', text)


    return text
```

+ Code + Markdown

[15]:
```
# Create a function to tokenize a set of texts
import emoji
import unicodedata
def preprocessing_for_bert(data, text_preprocessing_fn = text_preprocessing ):
    """Perform required preprocessing steps for pretrained BERT.
    @param    data (np.array): Array of texts to be processed.
    @return   input_ids (torch.Tensor): Tensor of token ids to be fed to a model.
    @return   attention_masks (torch.Tensor): Tensor of indices specifying which
                  tokens should be attended to by the model.
    """
    # Create empty lists to store outputs
    input_ids = []
    attention_masks = []
    tokenizer = AutoTokenizer.from_pretrained("bashar-talafha/multi-dialect-bert-base-arabic")

    # For every sentence...
    for i,sent in enumerate(data):
        # `encode_plus` will:
        #    (1) Tokenize the sentence
        #    (2) Add the `[CLS]` and `[SEP]` token to the start and end
        #    (3) Truncate/Pad sentence to max length
        #    (4) Map tokens to their IDs
        #    (5) Create attention mask
        #    (6) Return a dictionary of outputs
        encoded_sent = tokenizer.encode_plus(
            text=text_preprocessing_fn(sent),  # Preprocess sentence
            add_special_tokens=True,        # Add `[CLS]` and `[SEP]`
            max_length=MAX_LEN,                  # Max length to truncate/pad
            padding='max_length',        # Pad sentence to max length
            #return_tensors='pt',           # Return PyTorch tensor
            return_attention_mask=True,     # Return attention mask
            truncation = True
            )

        # Add the outputs to the lists
        input_ids.append(encoded_sent.get('input_ids'))
        attention_masks.append(encoded_sent.get('attention_mask'))
    # Convert lists to tensors
    input_ids = torch.tensor(input_ids)
    attention_masks = torch.tensor(attention_masks)

    return input_ids, attention_masks
```

+ Code + Markdown

[16]:

```python
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
le.fit(balanced_df.dialect)

y_train_labeled = le.transform(balanced_df.dialect)
balanced_df['labeled']=y_train_labeled
```

+ Code + Markdown

[17]:

```python
from sklearn.model_selection import train_test_split

X = balanced_df.text
y = balanced_df.labeled
x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=42,stratify=y,test_size=0.3)
X_train, X_val, y_train1, y_val = train_test_split(x_train, y_train, random_state=42,stratify=y_train,test_size=0.2)
```

[18]:

```python
import re
# Specify `MAX_LEN`
MAX_LEN =  280

# Print sentence 0 and its encoded token ids
token_ids = list(preprocessing_for_bert([X[0]])[0].squeeze().numpy())
print('Original: ', X[0])
print('Token IDs: ', token_ids)

# Run function `preprocessing_for_bert` on the train set and the validation set
print('Tokenizing data...')
train_inputs, train_masks = preprocessing_for_bert(X_train)
val_inputs, val_masks = preprocessing_for_bert(X_val)
```

```
Original:  @Usama_LY @MJJLegion قبل يمرن عليا تغريدات ليك تذكر فيه درتك من جمهوره حصل خير سوء فهم
Token IDs:  [2, 2158, 9549, 1022, 12204, 3584, 21507, 11176, 8083, 2369, 4392, 2398, 1762, 11242, 1033, 4759, 4455, 8548, 5109, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0]
Tokenizing data...
```

+ Code + Markdown

Create data loaders for test and validation sets

[19]:

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

# Convert other data types to torch.Tensor
train_labels = torch.tensor(y_train1.tolist())
val_labels = torch.tensor(y_val.tolist())

# For fine-tuning BERT, the authors recommend a batch size of 16 or 32.
batch_size = 32

# Create the DataLoader for our training set
train_data = TensorDataset(train_inputs, train_masks, train_labels)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler, batch_size=batch_size)

# Create the DataLoader for our validation set
val_data = TensorDataset(val_inputs, val_masks, val_labels)
val_sampler = SequentialSampler(val_data)
val_dataloader = DataLoader(val_data, sampler=val_sampler, batch_size=batch_size)
```

+ Code + Markdown

Define model initialization class and functions

[20]:

```python
%%time
import torch
import torch.nn as nn
from transformers import BertModel

# Create the BertClassfier class
class BertClassifier(nn.Module):
    """Bert Model for Classification Tasks.
    """
    def __init__(self, freeze_bert=False):
        """
        @param    bert: a BertModel object
        @param    classifier: a torch.nn.Module classifier
        @param    freeze_bert (bool): Set `False` to fine-tune the BERT model
        """
        super(BertClassifier, self).__init__()
        # Specify hidden size of BERT, hidden size of our classifier, and number of labels
        D_in =768
        H, D_out = 50, 18
```

```
        # Instantiate BERT model
        self.bert = AutoModel.from_pretrained("bashar-talafha/multi-dialect-bert-base-arabic")
        # Instantiate an one-layer feed-forward classifier
        self.classifier = nn.Sequential(
            nn.Linear(D_in, H),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(H, D_out)
        )

        # Freeze the BERT model
        if freeze_bert:
            for param in self.bert.parameters():
                param.requires_grad = False

    def forward(self, input_ids, attention_mask):
        """
        Feed input to BERT and the classifier to compute logits.
        @param    input_ids (torch.Tensor): an input tensor with shape (batch_size,
                      max_length)
        @param    attention_mask (torch.Tensor): a tensor that hold attention mask
                      information with shape (batch_size, max_length)
        @return   logits (torch.Tensor): an output tensor with shape (batch_size,
                      num_labels)
        """
        # Feed input to BERT
        outputs = self.bert(input_ids=input_ids,
                            attention_mask=attention_mask)

        # Extract the last hidden state of the token `[CLS]` for classification task
        last_hidden_state_cls = outputs[0][:, 0, :]

        # Feed input to classifier to compute logits
        logits = self.classifier(last_hidden_state_cls)

        return logits
```

```
CPU times: user 117 µs, sys: 3 µs, total: 120 µs
Wall time: 124 µs
```

+ Code + Markdown

[21]:

```
from transformers import AdamW, get_linear_schedule_with_warmup

from torch.optim import SparseAdam, Adam
def initialize_model(epochs=4):
    """Initialize the Bert Classifier, the optimizer and the learning rate scheduler.
    """
    # Instantiate Bert Classifier
    bert_classifier = BertClassifier(freeze_bert=False)
    # Tell PyTorch to run the model on GPU
    bert_classifier.to(device)

    # Create the optimizer
    optimizer = AdamW(params=list(bert_classifier.parameters()),
                      lr=5e-5,    # Default learning rate
                      eps=1e-8    # Default epsilon value
                      )

    # Total number of training steps
    total_steps = len(train_dataloader) * epochs

    # Set up the learning rate scheduler
    scheduler = get_linear_schedule_with_warmup(optimizer,
                                                num_warmup_steps=0, # Default value
                                                num_training_steps=total_steps)
    return bert_classifier, optimizer, scheduler
```

+ Code + Markdown

Define model train and evaluate functions

[22]:

```
import random
import time
import torch
import torch.nn as nn
# Specify loss function
loss_fn = nn.CrossEntropyLoss()

def set_seed(seed_value=42):
    """Set seed for reproducibility.
    """
    random.seed(seed_value)
    np.random.seed(seed_value)
    torch.manual_seed(seed_value)
    torch.cuda.manual_seed_all(seed_value)

def train(model, train_dataloader, val_dataloader=None, epochs=4, evaluation=False):
    """Train the BertClassifier model.
    """
    # Start training loop
    print("Start training...\n")
```

```
    print("Start training...\n")
    for epoch_i in range(epochs):
        # ========================================
        #               Training
        # ========================================
        # Print the header of the result table
        print(f"{'Epoch':^7} | {'Batch':^7} | {'Train Loss':^12} | {'Val Loss':^10} | {'Val Acc':^9} | {'Elapsed':^9}")
        print("-"*70)

        # Measure the elapsed time of each epoch
        t0_epoch, t0_batch = time.time(), time.time()

        # Reset tracking variables at the beginning of each epoch
        total_loss, batch_loss, batch_counts = 0, 0, 0

        # Put the model into the training mode
        model.train()

        # For each batch of training data...
        for step, batch in enumerate(train_dataloader):
            batch_counts +=1
            # Load batch to GPU
            b_input_ids, b_attn_mask, b_labels = tuple(t.to(device) for t in batch)

            # Zero out any previously calculated gradients
            model.zero_grad()

            # Perform a forward pass. This will return logits.
            logits = model(b_input_ids, b_attn_mask)

            # Compute loss and accumulate the loss values
            loss = loss_fn(logits, b_labels)
            batch_loss += loss.item()
            total_loss += loss.item()

            # Perform a backward pass to calculate gradients
            loss.backward()

            # Clip the norm of the gradients to 1.0 to prevent "exploding gradients"
            torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

            # Update parameters and the learning rate
            optimizer.step()
            scheduler.step()

            # Print the loss values and time elapsed for every 20 batches
            if (step % 20 == 0 and step != 0) or (step == len(train_dataloader) - 1):
                # Calculate time elapsed for 20 batches
                time_elapsed = time.time() - t0_batch

                # Print training results
                print(f"{epoch_i + 1:^7} | {step:^7} | {batch_loss / batch_counts:^12.6f} | {'-':^10} | {'-':^9} | {time_elapsed:^9.2f}")

                # Reset batch tracking variables
                batch_loss, batch_counts = 0, 0
                t0_batch = time.time()

        # Calculate the average loss over the entire training data
        avg_train_loss = total_loss / len(train_dataloader)

        print("-"*70)
        # ========================================
        #               Evaluation
        # ========================================
        if evaluation == True:
            # After the completion of each training epoch, measure the model's performance
            # on our validation set.
            val_loss, val_accuracy = evaluate(model, val_dataloader)

            # Print performance over the entire training data
            time_elapsed = time.time() - t0_epoch

            print(f"{epoch_i + 1:^7} | {'-':^7} | {avg_train_loss:^12.6f} | {val_loss:^10.6f} | {val_accuracy:^9.2f} | {time_elapsed:^9.2f}")
            print("-"*70)
        print("\n")

    print("Training complete!")


def evaluate(model, val_dataloader):
    """After the completion of each training epoch, measure the model's performance
    on our validation set.
    """
    # Put the model into the evaluation mode. The dropout layers are disabled during
    # the test time.
    model.eval()

    # Tracking variables
    val_accuracy = []
    val_loss = []

    # For each batch in our validation set...
    for batch in val_dataloader:
        # Load batch to GPU
        b_input_ids, b_attn_mask, b_labels = tuple(t.to(device) for t in batch)

        # Compute logits
```

```
        # Compute logits
        with torch.no_grad():
            logits = model(b_input_ids, b_attn_mask)

        # Compute loss
        loss = loss_fn(logits, b_labels)
        val_loss.append(loss.item())

        # Get the predictions
        preds = torch.argmax(logits, dim=1).flatten()

        # Calculate the accuracy rate
        accuracy = (preds == b_labels).cpu().numpy().mean() * 100
        val_accuracy.append(accuracy)

    # Compute the average accuracy and loss over the validation set.
    val_loss = np.mean(val_loss)
    val_accuracy = np.mean(val_accuracy)

    return val_loss, val_accuracy
```

Define prediction and test set evaluation functions

+ Code + Markdown

[23]:

```
import torch.nn.functional as F

def bert_predict(model, test_dataloader):
    """Perform a forward pass on the trained BERT model to predict probabilities
    on the test set.
    """
    # Put the model into the evaluation mode. The dropout layers are disabled during
    # the test time.
    model.eval()

    all_logits = []

    # For each batch in our test set...
    for batch in test_dataloader:
        # Load batch to GPU
        b_input_ids, b_attn_mask = tuple(t.to(device) for t in batch)[:2]

        # Compute logits
        with torch.no_grad():
            logits = model(b_input_ids, b_attn_mask)
        all_logits.append(logits)

    # Concatenate logits from each batch
    all_logits = torch.cat(all_logits, dim=0)

    # Apply softmax to calculate probabilities
    probs = F.softmax(all_logits, dim=1).cpu().numpy()

    return probs
```

+ Code + Markdown

[24]:

```
from sklearn import metrics


def evaluate_roc(probs, y_true):
    """
    - Print AUC and accuracy on the test set
    - Plot ROC
    @params    probs (np.array): an array of predicted probabilities with shape (len(y_true), 18)
    @params    y_true (np.array): an array of the true values with shape (len(y_true),)
    """

    # Get accuracy over the test set
    y_pre=[]
    for y in probs:
        y_pre.append(np.argmax(y))

    y_true_inverse=list(le.inverse_transform(y_true))
    y_pre_inverse=list(le.inverse_transform(y_pre))
    print(metrics.classification_report(y_true_inverse, y_pre_inverse))
```

+ Code + Markdown

Initialize and train model

[25]:

```
import numpy as np
set_seed(42)
bert_classifier, optimizer, scheduler = initialize_model(epochs=2)
train(bert_classifier, train_dataloader, val_dataloader, epochs=2, evaluation=True)
```

```
Downloading: 100% 445M/445M [00:13<00:00, 35.7MB/s]
```

```
Start training...

 Epoch  |  Batch  |  Train Loss  |  Val Loss  |  Val Acc  |  Elapsed
----------------------------------------------------------------------
   1    |   20    |   2.905412   |     -      |     -     |   18.88
   1    |   40    |   2.792159   |     -      |     -     |   17.26
   1    |   60    |   2.674144   |     -      |     -     |   17.26
   1    |   80    |   2.592439   |     -      |     -     |   17.28
   1    |  100    |   2.592987   |     -      |     -     |   17.24
   1    |  120    |   2.548039   |     -      |     -     |   17.26
   1    |  140    |   2.545216   |     -      |     -     |   17.27
   1    |  160    |   2.487133   |     -      |     -     |   17.25
   1    |  180    |   2.493343   |     -      |     -     |   17.26
   1    |  200    |   2.541870   |     -      |     -     |   17.27
   1    |  220    |   2.478550   |     -      |     -     |   17.25
   1    |  240    |   2.397983   |     -      |     -     |   17.30
   1    |  260    |   2.532803   |     -      |     -     |   17.30
   1    |  280    |   2.458188   |     -      |     -     |   17.32
   1    |  300    |   2.368405   |     -      |     -     |   17.20
   1    |  320    |   2.481451   |     -      |     -     |   17.33
   1    |  340    |   2.424271   |     -      |     -     |   17.25
   1    |  360    |   2.372714   |     -      |     -     |   17.30
   1    |  380    |   2.415003   |     -      |     -     |   17.24
   1    |  400    |   2.368171   |     -      |     -     |   17.34
   1    |  420    |   2.448161   |     -      |     -     |   17.31
   1    |  440    |   2.291367   |     -      |     -     |   17.23
   1    |  460    |   2.315570   |     -      |     -     |   17.42
   1    |  480    |   2.374227   |     -      |     -     |   17.23
   1    |  500    |   2.387413   |     -      |     -     |   17.30
   1    |  520    |   2.355588   |     -      |     -     |   17.27
   1    |  540    |   2.303403   |     -      |     -     |   17.28
   1    |  560    |   2.263368   |     -      |     -     |   17.26
   1    |  580    |   2.247942   |     -      |     -     |   17.30
   1    |  600    |   2.207756   |     -      |     -     |   17.29
   1    |  620    |   2.352938   |     -      |     -     |   17.26
   1    |  640    |   2.328185   |     -      |     -     |   17.25
   1    |  660    |   2.302134   |     -      |     -     |   17.25
   1    |  680    |   2.270167   |     -      |     -     |   17.23
   1    |  700    |   2.267692   |     -      |     -     |   17.31
   1    |  720    |   2.226176   |     -      |     -     |   17.28
   1    |  740    |   2.175682   |     -      |     -     |   17.24
   1    |  760    |   2.256877   |     -      |     -     |   17.26
   1    |  780    |   2.213803   |     -      |     -     |   17.22
   1    |  800    |   2.194803   |     -      |     -     |   17.27
   1    |  820    |   2.171709   |     -      |     -     |   17.25
   1    |  840    |   2.159387   |     -      |     -     |   17.28
   1    |  860    |   2.249438   |     -      |     -     |   17.24
   1    |  880    |   2.243803   |     -      |     -     |   17.29
   1    |  900    |   2.130780   |     -      |     -     |   17.28
   1    |  920    |   2.157652   |     -      |     -     |   17.30
   1    |  940    |   2.152799   |     -      |     -     |   17.24
   1    |  960    |   2.192036   |     -      |     -     |   17.28
   1    |  980    |   2.136584   |     -      |     -     |   17.26
   1    | 1000    |   2.178348   |     -      |     -     |   17.29
   1    | 1020    |   2.202164   |     -      |     -     |   17.26
   1    | 1040    |   2.111402   |     -      |     -     |   17.26
   1    | 1060    |   2.195876   |     -      |     -     |   17.38
   1    | 1080    |   2.218665   |     -      |     -     |   17.29
   1    | 1100    |   2.155673   |     -      |     -     |   17.33
   1    | 1120    |   2.061243   |     -      |     -     |   17.25
   1    | 1140    |   2.135322   |     -      |     -     |   17.31
   1    | 1160    |   2.090140   |     -      |     -     |   17.26
   1    | 1180    |   2.150988   |     -      |     -     |   17.33
   1    | 1200    |   2.068841   |     -      |     -     |   17.25
   1    | 1220    |   2.069673   |     -      |     -     |   17.31
   1    | 1240    |   2.082652   |     -      |     -     |   17.26
   1    | 1260    |   2.076406   |     -      |     -     |   17.29
   1    | 1280    |   2.143270   |     -      |     -     |   17.25
   1    | 1300    |   2.186998   |     -      |     -     |   17.26
   1    | 1320    |   2.174559   |     -      |     -     |   17.30
   1    | 1340    |   2.150315   |     -      |     -     |   17.26
   1    | 1360    |   1.964957   |     -      |     -     |   17.31
   1    | 1380    |   2.116152   |     -      |     -     |   17.28
   1    | 1400    |   2.047488   |     -      |     -     |   17.30
   1    | 1420    |   2.011968   |     -      |     -     |   17.25
   1    | 1440    |   2.062179   |     -      |     -     |   17.33
   1    | 1460    |   2.068820   |     -      |     -     |   17.26
   1    | 1480    |   2.116566   |     -      |     -     |   17.27
   1    | 1500    |   2.050876   |     -      |     -     |   17.28
   1    | 1520    |   2.145624   |     -      |     -     |   17.32
   1    | 1540    |   2.040804   |     -      |     -     |   17.26
   1    | 1560    |   2.095480   |     -      |     -     |   17.32
   1    | 1580    |   2.125778   |     -      |     -     |   17.22
   1    | 1600    |   2.108101   |     -      |     -     |   17.26
   1    | 1620    |   2.135203   |     -      |     -     |   17.29
   1    | 1640    |   2.051617   |     -      |     -     |   17.25
   1    | 1660    |   2.086854   |     -      |     -     |   17.42
   1    | 1680    |   2.047755   |     -      |     -     |   17.27
   1    | 1700    |   2.008897   |     -      |     -     |   17.36
   1    | 1720    |   2.099893   |     -      |     -     |   17.27
   1    | 1740    |   1.954672   |     -      |     -     |   17.41
   1    | 1760    |   2.027194   |     -      |     -     |   17.23
   1    | 1780    |   2.052625   |     -      |     -     |   17.30
   1    | 1800    |   1.989217   |     -      |     -     |   17.26
   1    | 1820    |   2.024528   |     -      |     -     |   17.34
   1    | 1840    |   2.060102   |     -      |     -     |   17.25
   1    | 1860    |   2.009275   |     -      |     -     |   17.30
   1    | 1880    |   1.989793   |     -      |     -     |   17.29
   1    | 1900    |   1.989617   |     -      |     -     |   17.23
   1    | 1920    |   2.028004   |     -      |     -     |   17.29
   1    | 1940    |   1.957188   |     -      |     -     |   17.24
   1    | 1960    |   2.007984   |     -      |     -     |   17.31
   1    | 1980    |   1.923737   |     -      |     -     |   17.28
   1    | 2000    |   2.067610   |     -      |     -     |   17.30
   1    | 2020    |   1.918995   |     -      |     -     |   17.26
   1    | 2040    |   1.963863   |     -      |     -     |   17.28
   1    | 2060    |   2.017803   |     -      |     -     |   17.29
   1    | 2080    |   1.863943   |     -      |     -     |   17.31
   1    | 2100    |   1.938618   |     -      |     -     |   17.25
   1    | 2120    |   1.897562   |     -      |     -     |   17.31
   1    | 2140    |   1.962358   |     -      |     -     |   17.27
   1    | 2160    |   1.995835   |     -      |     -     |   17.25
   1    | 2180    |   1.958068   |     -      |     -     |   17.29
   1    | 2200    |   1.998564   |     -      |     -     |   17.31
   1    | 2220    |   1.922862   |     -      |     -     |   17.28
   1    | 2240    |   2.027766   |     -      |     -     |   17.27
   1    | 2260    |   1.981937   |     -      |     -     |   17.32
   1    | 2280    |   1.988274   |     -      |     -     |   17.24
   1    | 2300    |   2.009622   |     -      |     -     |   17.29
   1    | 2320    |   1.956177   |     -      |     -     |   17.23
   1    | 2340    |   1.900466   |     -      |     -     |   17.27
   1    | 2360    |   1.958575   |     -      |     -     |   17.27
   1    | 2380    |   1.997697   |     -      |     -     |   17.30
   1    | 2400    |   1.861057   |     -      |     -     |   17.26
   1    | 2420    |   1.887357   |     -      |     -     |   17.42
   1    | 2440    |   1.885029   |     -      |     -     |   17.28
```

1	2460	1.958453	-	-	17.33
1	2480	1.952826	-	-	17.25
1	2500	1.879862	-	-	17.29
1	2520	2.023129	-	-	17.24
1	2540	1.991635	-	-	17.27
1	2560	1.930353	-	-	17.25
1	2580	1.985340	-	-	17.29
1	2600	1.879155	-	-	17.26
1	2620	1.868723	-	-	17.28
1	2640	1.947517	-	-	17.30
1	2660	1.984545	-	-	17.30
1	2680	1.919550	-	-	17.27
1	2700	1.866402	-	-	17.22
1	2720	1.962673	-	-	17.29
1	2740	1.959405	-	-	17.27
1	2760	1.910125	-	-	17.28
1	2780	1.945682	-	-	17.29
1	2800	1.910555	-	-	17.37
1	2820	1.981359	-	-	17.23
1	2840	1.946151	-	-	17.33
1	2860	1.836785	-	-	17.25
1	2880	1.849760	-	-	17.30
1	2900	1.935954	-	-	17.29
1	2920	1.868626	-	-	17.31
1	2940	1.848168	-	-	17.24
1	2960	1.930525	-	-	17.32
1	2980	1.865083	-	-	17.29
1	3000	1.941891	-	-	17.26
1	3020	1.975532	-	-	17.30
1	3040	1.874161	-	-	17.26
1	3060	1.906676	-	-	17.30
1	3080	1.793020	-	-	17.31
1	3100	1.891247	-	-	17.28
1	3120	1.953954	-	-	17.26
1	3140	1.888471	-	-	17.29
1	3160	1.899189	-	-	17.25
1	3180	1.812869	-	-	17.29
1	3200	1.736656	-	-	17.29
1	3220	1.864503	-	-	17.35
1	3240	1.870227	-	-	17.28
1	3260	1.877983	-	-	17.29
1	3280	1.759563	-	-	17.27
1	3300	1.829786	-	-	17.30
1	3320	1.763993	-	-	17.33
1	3340	1.829164	-	-	17.30
1	3360	1.842772	-	-	17.27
1	3380	1.870119	-	-	17.30
1	3400	1.868172	-	-	17.25
1	3420	1.756502	-	-	17.29
1	3440	1.840756	-	-	17.28
1	3460	1.972985	-	-	17.29
1	3480	1.902414	-	-	17.35
1	3500	1.928113	-	-	17.26
1	3520	1.809245	-	-	17.30
1	3540	1.857309	-	-	17.27
1	3560	1.777079	-	-	17.32
1	3580	1.692111	-	-	17.27
1	3600	1.815120	-	-	17.31
1	3620	1.748717	-	-	17.26
1	3640	1.753332	-	-	17.29
1	3660	1.779312	-	-	17.27
1	3680	1.872937	-	-	17.34
1	3700	1.741115	-	-	17.25
1	3720	1.831883	-	-	17.37
1	3740	1.860496	-	-	17.25
1	3760	1.908190	-	-	17.29
1	3780	1.855175	-	-	17.30
1	3800	1.855351	-	-	17.32
1	3820	1.882048	-	-	17.27
1	3840	1.905763	-	-	17.31
1	3860	1.838924	-	-	17.26
1	3880	1.691092	-	-	17.26
1	3900	1.826111	-	-	17.29
1	3920	1.729100	-	-	17.28
1	3940	1.869449	-	-	17.29
1	3960	1.792674	-	-	17.27
1	3980	1.758597	-	-	17.31
1	4000	1.827846	-	-	17.26
1	4020	1.692336	-	-	17.34
1	4040	1.774501	-	-	17.26
1	4060	1.827992	-	-	17.31
1	4080	1.789958	-	-	17.26
1	4100	1.815415	-	-	17.32
1	4120	1.763962	-	-	17.27
1	4140	1.766313	-	-	17.29
1	4160	1.885922	-	-	17.28
1	4180	1.743453	-	-	17.28
1	4200	1.767985	-	-	17.34
1	4220	1.759504	-	-	17.27
1	4240	1.751052	-	-	17.30
1	4260	1.643862	-	-	17.27
1	4280	1.894516	-	-	17.31
1	4300	1.799803	-	-	17.29
1	4320	1.750824	-	-	17.28
1	4340	1.772157	-	-	17.28
1	4360	1.730833	-	-	17.30
1	4380	1.836523	-	-	17.30
1	4400	1.754783	-	-	17.28
1	4420	1.762946	-	-	17.31
1	4440	1.706747	-	-	17.30
1	4460	1.774028	-	-	17.25
1	4480	1.849470	-	-	17.32
1	4500	1.758185	-	-	17.26
1	4520	1.663000	-	-	17.34
1	4540	1.731897	-	-	17.27
1	4560	1.882221	-	-	17.31
1	4580	1.847148	-	-	17.28
1	4600	1.886169	-	-	17.32
1	4620	1.815805	-	-	17.30
1	4640	1.713586	-	-	17.28
1	4660	1.812896	-	-	17.30
1	4680	1.719371	-	-	17.25
1	4700	1.926937	-	-	17.29
1	4720	1.706187	-	-	17.27
1	4740	1.706609	-	-	17.29
1	4760	1.826877	-	-	17.28
1	4780	1.630427	-	-	17.26
1	4800	1.657475	-	-	17.27
1	4820	1.678020	-	-	17.29
1	4840	1.756714	-	-	17.26
1	4860	1.746957	-	-	17.31
1	4880	1.758651	-	-	17.24
1	4900	1.746250	-	-	17.32
1	4920	1.787137	-	-	17.26
1	4940	1.739552	-	-	17.33
1	4960	1.754899	-	-	17.28

1	4980	1.750225	-	-	17.36
1	5000	1.746031	-	-	17.26
1	5020	1.654127	-	-	17.34
1	5040	1.753785	-	-	17.29
1	5060	1.732476	-	-	17.30
1	5080	1.758348	-	-	17.29
1	5100	1.732088	-	-	17.25
1	5120	1.683039	-	-	17.31
1	5140	1.640774	-	-	17.25
1	5160	1.783431	-	-	17.29
1	5180	1.748950	-	-	17.27
1	5200	1.640448	-	-	17.28
1	5220	1.670879	-	-	17.28
1	5240	1.680723	-	-	17.28
1	5260	1.699317	-	-	17.29
1	5280	1.729701	-	-	17.29
1	5300	1.839186	-	-	17.25
1	5320	1.766451	-	-	17.33
1	5340	1.710037	-	-	17.27
1	5360	1.758746	-	-	17.34
1	5380	1.743721	-	-	17.26
1	5400	1.768446	-	-	17.31
1	5420	1.651699	-	-	17.27
1	5440	1.724679	-	-	17.33
1	5460	1.638089	-	-	17.31
1	5480	1.771385	-	-	17.24
1	5500	1.705908	-	-	17.28
1	5520	1.679745	-	-	17.28
1	5540	1.684438	-	-	17.29
1	5560	1.739460	-	-	17.29
1	5580	1.683771	-	-	17.36
1	5600	1.697900	-	-	17.30
1	5620	1.595439	-	-	17.29
1	5640	1.729463	-	-	17.30
1	5660	1.730659	-	-	17.28
1	5680	1.735906	-	-	17.32
1	5700	1.712958	-	-	17.26
1	5720	1.794305	-	-	17.29
1	5740	1.731807	-	-	17.28
1	5760	1.627953	-	-	17.24
1	5780	1.682997	-	-	17.31
1	5800	1.593516	-	-	17.25
1	5820	1.620961	-	-	17.34
1	5840	1.815506	-	-	17.26
1	5860	1.668801	-	-	17.34
1	5880	1.760497	-	-	17.26
1	5900	1.732579	-	-	17.32
1	5920	1.620013	-	-	17.29
1	5940	1.696644	-	-	17.27
1	5960	1.711029	-	-	17.30
1	5980	1.622642	-	-	17.26
1	6000	1.833056	-	-	17.34
1	6020	1.636639	-	-	17.27
1	6040	1.658209	-	-	17.31
1	6060	1.791460	-	-	17.29
1	6080	1.683944	-	-	17.28
1	6100	1.586388	-	-	17.33
1	6120	1.602808	-	-	17.26
1	6140	1.672122	-	-	17.33
1	6160	1.609057	-	-	17.31
1	6180	1.578898	-	-	17.27
1	6200	1.654945	-	-	17.27
1	6220	1.643757	-	-	17.27
1	6240	1.618143	-	-	17.34
1	6260	1.745799	-	-	17.26
1	6280	1.684796	-	-	17.32
1	6300	1.626911	-	-	17.27
1	6320	1.768206	-	-	17.31
1	6340	1.549539	-	-	17.28
1	6360	1.657790	-	-	17.34
1	6380	1.673008	-	-	17.29
1	6400	1.609922	-	-	17.27
1	6420	1.671912	-	-	17.34
1	6440	1.661268	-	-	17.28
1	6460	1.695318	-	-	17.30
1	6480	1.641971	-	-	17.28
1	6500	1.573102	-	-	17.35
1	6520	1.605884	-	-	17.27
1	6540	1.667368	-	-	17.30
1	6560	1.635643	-	-	17.31
1	6580	1.635694	-	-	17.33
1	6600	1.617396	-	-	17.27
1	6620	1.677038	-	-	17.34
1	6660	1.665455	-	-	17.31
1	6680	1.626494	-	-	17.29
1	6700	1.633534	-	-	17.29
1	6720	1.631523	-	-	17.27
1	6740	1.702803	-	-	17.28
1	6760	1.614026	-	-	17.24
1	6780	1.596978	-	-	17.29
1	6800	1.694792	-	-	17.30
1	6820	1.564003	-	-	17.35
1	6840	1.632314	-	-	17.33
1	6900	1.648939	-	-	17.29
1	6920	1.625832	-	-	17.30
1	6940	1.524428	-	-	17.33
1	6960	1.632459	-	-	17.31
1	6980	1.620285	-	-	17.26
1	7000	1.613221	-	-	17.31
1	7020	1.601944	-	-	17.27
1	7040	1.620317	-	-	17.31
1	7060	1.517082	-	-	17.28
1	7080	1.651792	-	-	17.35
1	7100	1.493315	-	-	17.26
1	7120	1.633968	-	-	17.37
1	7140	1.630416	-	-	17.27
1	7160	1.659135	-	-	17.30
1	7180	1.564413	-	-	17.31
1	7200	1.583826	-	-	17.27
1	7220	1.620240	-	-	17.31
1	7240	1.642525	-	-	17.30
1	7260	1.512139	-	-	17.34
1	7280	1.674114	-	-	17.27
1	7300	1.568512	-	-	17.32
1	7320	1.550349	-	-	17.31
1	7340	1.570267	-	-	17.28
1	7360	1.546832	-	-	17.28
1	7380	1.582105	-	-	17.30
1	7400	1.546930	-	-	17.27
1	7420	1.707374	-	-	17.29
1	7440	1.570675	-	-	17.30
1	7460	1.598672	-	-	17.32
1	7480	1.595873	-	-	17.28
1	7500	1.570185	-	-	17.32
1	7520	1.543380	-	-	17.28
1	7540	1.559987	-	-	17.34
1	7560	1.549783	-	-	17.26

1	7580	1.531683	-	-	17.32
1	7600	1.514270	-	-	17.27
1	7620	1.619528	-	-	17.33
1	7640	1.684243	-	-	17.33
1	7660	1.633290	-	-	17.29
1	7680	1.592648	-	-	17.31
1	7700	1.638528	-	-	17.29
1	7720	1.576524	-	-	17.29
1	7740	1.676247	-	-	17.27
1	7760	1.608278	-	-	17.32
1	7780	1.591808	-	-	17.27
1	7800	1.570812	-	-	17.31
1	7820	1.508301	-	-	17.29
1	7840	1.585828	-	-	17.34
1	7860	1.538568	-	-	17.27
1	7874	1.542294	-	-	12.13
1	-	1.869310	1.391417	57.59	7360.52

Epoch	Batch	Train Loss	Val Loss	Val Acc	Elapsed
2	20	1.452836	-	-	18.20
2	40	1.367492	-	-	17.30
2	60	1.234191	-	-	17.23
2	80	1.453893	-	-	17.36
2	100	1.385806	-	-	17.28
2	120	1.460659	-	-	17.31
2	140	1.344508	-	-	17.27
2	160	1.388527	-	-	17.30
2	180	1.497710	-	-	17.31
2	200	1.457584	-	-	17.32
2	220	1.439766	-	-	17.32
2	240	1.284731	-	-	17.29
2	260	1.397234	-	-	17.27
2	280	1.402668	-	-	17.32
2	300	1.298658	-	-	17.29
2	320	1.320876	-	-	17.30
2	340	1.325723	-	-	17.27
2	360	1.403274	-	-	17.33
2	380	1.421736	-	-	17.29
2	400	1.419941	-	-	17.30
2	420	1.435112	-	-	17.29
2	440	1.486431	-	-	17.25
2	460	1.349937	-	-	17.30
2	480	1.398613	-	-	17.27
2	500	1.341464	-	-	17.34
2	520	1.447253	-	-	17.29
2	540	1.359857	-	-	17.32
2	560	1.360780	-	-	17.28
2	580	1.398242	-	-	17.27
2	600	1.381850	-	-	17.25
2	620	1.441915	-	-	17.29
2	640	1.409312	-	-	17.28
2	660	1.311477	-	-	17.35
2	680	1.304744	-	-	17.27
2	700	1.497247	-	-	17.27
2	720	1.411823	-	-	17.25
2	740	1.302560	-	-	17.29
2	760	1.338967	-	-	17.24
2	780	1.425289	-	-	17.28
2	800	1.348859	-	-	17.25
2	820	1.315503	-	-	17.29
2	840	1.366091	-	-	17.31
2	860	1.251855	-	-	17.25
2	880	1.406343	-	-	17.28
2	900	1.351027	-	-	17.29
2	920	1.354221	-	-	17.28
2	940	1.324943	-	-	17.26
2	960	1.365832	-	-	17.34
2	980	1.484821	-	-	17.25
2	1000	1.476518	-	-	17.30
2	1020	1.354045	-	-	17.25
2	1040	1.343362	-	-	17.34
2	1060	1.387431	-	-	17.27
2	1080	1.397028	-	-	17.28
2	1100	1.284730	-	-	17.26
2	1120	1.346806	-	-	17.29
2	1140	1.337865	-	-	17.29
2	1160	1.299708	-	-	17.29
2	1180	1.412604	-	-	17.29
2	1200	1.341235	-	-	17.31
2	1220	1.345142	-	-	17.26
2	1240	1.372086	-	-	17.32
2	1260	1.399431	-	-	17.26
2	1280	1.389397	-	-	17.29
2	1300	1.336093	-	-	17.30
2	1320	1.368670	-	-	17.28
2	1340	1.294485	-	-	17.30
2	1360	1.383885	-	-	17.26
2	1380	1.341616	-	-	17.31
2	1400	1.327378	-	-	17.25
2	1420	1.317489	-	-	17.32
2	1440	1.370308	-	-	17.25
2	1460	1.288944	-	-	17.39
2	1480	1.376357	-	-	17.25
2	1500	1.329665	-	-	17.28
2	1520	1.423167	-	-	17.27
2	1540	1.318122	-	-	17.32
2	1560	1.317061	-	-	17.27
2	1580	1.403555	-	-	17.26
2	1600	1.320944	-	-	17.28
2	1620	1.382975	-	-	17.29
2	1640	1.330467	-	-	17.25
2	1660	1.379438	-	-	17.30
2	1680	1.352685	-	-	17.25
2	1700	1.315578	-	-	17.26
2	1720	1.196152	-	-	17.27
2	1740	1.370636	-	-	17.26
2	1760	1.314822	-	-	17.32
2	1780	1.340521	-	-	17.28
2	1800	1.356131	-	-	17.31
2	1820	1.345140	-	-	17.26
2	1840	1.348033	-	-	17.28
2	1860	1.362504	-	-	17.27
2	1880	1.411544	-	-	17.25
2	1900	1.312775	-	-	17.29
2	1920	1.396535	-	-	17.33
2	1940	1.365701	-	-	17.27
2	1960	1.371671	-	-	17.31
2	1980	1.355399	-	-	17.29
2	2000	1.417872	-	-	17.30
2	2020	1.333440	-	-	17.25
2	2040	1.405396	-	-	17.33
2	2060	1.377791	-	-	17.28

2	2080	1.311804	-	-	17.28
2	2100	1.348507	-	-	17.27
2	2120	1.289452	-	-	17.26
2	2140	1.369431	-	-	17.32
2	2160	1.329592	-	-	17.25
2	2180	1.351485	-	-	17.36
2	2200	1.313691	-	-	17.29
2	2220	1.400217	-	-	17.27
2	2240	1.340524	-	-	17.25
2	2260	1.262663	-	-	17.28
2	2280	1.373999	-	-	17.24
2	2300	1.250139	-	-	17.30
2	2320	1.443443	-	-	17.28
2	2340	1.350526	-	-	17.32
2	2360	1.342641	-	-	17.25
2	2380	1.292064	-	-	17.26
2	2400	1.297370	-	-	17.29
2	2420	1.297621	-	-	17.32
2	2440	1.230482	-	-	17.25
2	2460	1.308652	-	-	17.32
2	2480	1.174223	-	-	17.28
2	2500	1.243408	-	-	17.32
2	2520	1.215985	-	-	17.27
2	2540	1.252894	-	-	17.27
2	2560	1.434580	-	-	17.35
2	2580	1.307588	-	-	17.29
2	2600	1.304248	-	-	17.33
2	2620	1.337573	-	-	17.27
2	2640	1.349692	-	-	17.29
2	2660	1.308986	-	-	17.24
2	2680	1.376505	-	-	17.30
2	2700	1.320605	-	-	17.26
2	2720	1.297319	-	-	17.32
2	2740	1.321357	-	-	17.28
2	2760	1.254447	-	-	17.30
2	2780	1.216975	-	-	17.29
2	2800	1.248111	-	-	17.26
2	2820	1.305802	-	-	17.26
2	2840	1.330968	-	-	17.28
2	2860	1.291345	-	-	17.24
2	2880	1.307895	-	-	17.30
2	2900	1.420936	-	-	17.26
2	2920	1.314293	-	-	17.30
2	2940	1.408788	-	-	17.31
2	2960	1.299001	-	-	17.30
2	2980	1.252357	-	-	17.35
2	3000	1.333855	-	-	17.29
2	3020	1.306855	-	-	17.28
2	3040	1.283469	-	-	17.27
2	3060	1.312660	-	-	17.28
2	3080	1.294601	-	-	17.24
2	3100	1.257382	-	-	17.30
2	3120	1.204168	-	-	17.28
2	3140	1.229914	-	-	17.26
2	3160	1.315226	-	-	17.29
2	3180	1.316215	-	-	17.31
2	3200	1.211031	-	-	17.27
2	3220	1.232763	-	-	17.29
2	3240	1.262001	-	-	17.28
2	3260	1.293366	-	-	17.28
2	3280	1.363591	-	-	17.25
2	3300	1.247784	-	-	17.33
2	3320	1.260368	-	-	17.30
2	3340	1.359466	-	-	17.30
2	3360	1.247201	-	-	17.28
2	3380	1.319397	-	-	17.29
2	3400	1.315022	-	-	17.31
2	3420	1.291927	-	-	17.27
2	3440	1.344708	-	-	17.29
2	3460	1.230427	-	-	17.28
2	3480	1.179014	-	-	17.31
2	3500	1.326856	-	-	17.30
2	3520	1.317235	-	-	17.29
2	3540	1.221582	-	-	17.27
2	3560	1.304064	-	-	17.31
2	3580	1.243939	-	-	17.30
2	3600	1.340795	-	-	17.28
2	3620	1.343277	-	-	17.26
2	3640	1.260964	-	-	17.33
2	3660	1.245085	-	-	17.26
2	3680	1.246595	-	-	17.28
2	3700	1.220748	-	-	17.30
2	3720	1.276096	-	-	17.28
2	3740	1.329601	-	-	17.27
2	3760	1.409517	-	-	17.30
2	3780	1.275684	-	-	17.24
2	3800	1.406963	-	-	17.27
2	3820	1.266252	-	-	17.27
2	3840	1.307640	-	-	17.24
2	3860	1.200176	-	-	17.30
2	3880	1.298850	-	-	17.24
2	3900	1.260903	-	-	17.31
2	3920	1.311465	-	-	17.25
2	3940	1.330672	-	-	17.28
2	3960	1.294953	-	-	17.25
2	3980	1.387147	-	-	17.25
2	4000	1.369593	-	-	17.26
2	4020	1.299095	-	-	17.26
2	4040	1.310053	-	-	17.27
2	4060	1.302136	-	-	17.28
2	4080	1.216719	-	-	17.23
2	4100	1.302149	-	-	17.34
2	4120	1.349074	-	-	17.22
2	4140	1.278372	-	-	17.30
2	4160	1.240909	-	-	17.24
2	4180	1.281254	-	-	17.33
2	4200	1.222808	-	-	17.27
2	4220	1.301983	-	-	17.25
2	4240	1.472294	-	-	17.33
2	4260	1.217858	-	-	17.26
2	4280	1.338966	-	-	17.32
2	4300	1.281877	-	-	17.25
2	4320	1.306368	-	-	17.27
2	4340	1.276634	-	-	17.30
2	4360	1.239443	-	-	17.25
2	4380	1.185364	-	-	17.30
2	4400	1.271504	-	-	17.29
2	4420	1.271121	-	-	17.26
2	4440	1.246219	-	-	17.31
2	4460	1.335925	-	-	17.22
2	4480	1.250305	-	-	17.30
2	4500	1.200029	-	-	17.26
2	4520	1.322196	-	-	17.29
2	4540	1.284718	-	-	17.24
2	4560	1.175020	-	-	17.30
2	4580	1.197722	-	-	17.25

2	4600	1.218318	-	-	17.28
2	4620	1.212036	-	-	17.31
2	4640	1.221129	-	-	17.29
2	4660	1.269433	-	-	17.28
2	4680	1.223502	-	-	17.24
2	4700	1.312585	-	-	17.26
2	4720	1.140421	-	-	17.26
2	4740	1.237469	-	-	17.29
2	4760	1.376269	-	-	17.27
2	4780	1.182695	-	-	17.28
2	4800	1.226235	-	-	17.29
2	4820	1.293165	-	-	17.26
2	4840	1.308365	-	-	17.26
2	4860	1.301679	-	-	17.32
2	4880	1.260025	-	-	17.26
2	4900	1.250501	-	-	17.29
2	4920	1.244637	-	-	17.26
2	4940	1.178177	-	-	17.32
2	4960	1.222066	-	-	17.27
2	4980	1.183351	-	-	17.33
2	5000	1.180165	-	-	17.27
2	5020	1.323332	-	-	17.31
2	5040	1.204218	-	-	17.30
2	5060	1.307882	-	-	17.25
2	5080	1.192056	-	-	17.30
2	5100	1.365472	-	-	17.25
2	5120	1.256273	-	-	17.30
2	5140	1.261899	-	-	17.28
2	5160	1.234313	-	-	17.28
2	5180	1.227965	-	-	17.26
2	5200	1.305676	-	-	17.27
2	5220	1.250717	-	-	17.24
2	5240	1.313911	-	-	17.25
2	5260	1.317809	-	-	17.25
2	5280	1.275526	-	-	17.26
2	5300	1.293145	-	-	17.27
2	5320	1.232843	-	-	17.33
2	5340	1.242084	-	-	17.24
2	5360	1.283650	-	-	17.28
2	5380	1.121857	-	-	17.31
2	5400	1.226691	-	-	17.26
2	5420	1.275475	-	-	17.25
2	5440	1.273362	-	-	17.28
2	5460	1.259681	-	-	17.28
2	5480	1.201260	-	-	17.25
2	5500	1.220362	-	-	17.30
2	5520	1.350287	-	-	17.24
2	5540	1.267945	-	-	17.35
2	5560	1.304840	-	-	17.24
2	5580	1.187211	-	-	17.32
2	5600	1.190507	-	-	17.24
2	5620	1.318739	-	-	17.30
2	5640	1.220356	-	-	17.26
2	5660	1.172965	-	-	17.28
2	5680	1.328928	-	-	17.30
2	5700	1.192968	-	-	17.25
2	5720	1.210772	-	-	17.31
2	5740	1.239988	-	-	17.28
2	5760	1.156542	-	-	17.30
2	5780	1.350058	-	-	17.27
2	5800	1.267592	-	-	17.25
2	5820	1.247361	-	-	17.28
2	5840	1.273874	-	-	17.35
2	5860	1.219724	-	-	17.30
2	5880	1.322122	-	-	17.39
2	5900	1.223195	-	-	17.26
2	5920	1.291344	-	-	17.41
2	5940	1.156140	-	-	17.25
2	5960	1.307187	-	-	17.40
2	5980	1.184727	-	-	17.30
2	6000	1.258419	-	-	17.35
2	6020	1.190671	-	-	17.36
2	6040	1.330795	-	-	17.28
2	6060	1.220339	-	-	17.30
2	6080	1.229620	-	-	17.27
2	6100	1.216548	-	-	17.30
2	6120	1.256790	-	-	17.26
2	6140	1.293780	-	-	17.30
2	6160	1.204008	-	-	17.28
2	6180	1.218000	-	-	17.31
2	6200	1.196684	-	-	17.27
2	6220	1.321837	-	-	17.27
2	6240	1.293977	-	-	17.26
2	6260	1.104884	-	-	17.27
2	6280	1.166340	-	-	17.26
2	6300	1.284924	-	-	17.33
2	6320	1.238602	-	-	17.25
2	6340	1.129915	-	-	17.32
2	6360	1.186137	-	-	17.24
2	6380	1.241087	-	-	17.32
2	6400	1.294953	-	-	17.27
2	6420	1.263963	-	-	17.26
2	6440	1.288098	-	-	17.33
2	6460	1.255804	-	-	17.27
2	6480	1.248296	-	-	17.32
2	6500	1.213451	-	-	17.25
2	6520	1.251008	-	-	17.26
2	6540	1.248209	-	-	17.27
2	6560	1.225394	-	-	17.30
2	6580	1.202930	-	-	17.27
2	6600	1.186384	-	-	17.27
2	6620	1.233510	-	-	17.28
2	6640	1.269770	-	-	17.26
2	6660	1.110629	-	-	17.28
2	6680	1.248995	-	-	17.28
2	6700	1.265441	-	-	17.27
2	6720	1.230369	-	-	17.38
2	6740	1.153920	-	-	17.27
2	6760	1.194965	-	-	17.39
2	6780	1.240001	-	-	17.28
2	6800	1.119166	-	-	17.41
2	6820	1.246261	-	-	17.25
2	6840	1.214320	-	-	17.39
2	6860	1.226695	-	-	17.36
2	6880	1.304469	-	-	17.20
2	6900	1.156275	-	-	17.35
2	6920	1.334242	-	-	17.23
2	6940	1.221179	-	-	17.29
2	6960	1.307091	-	-	17.28
2	6980	1.259201	-	-	17.32
2	7000	1.184664	-	-	17.28
2	7020	1.270479	-	-	17.26
2	7040	1.226489	-	-	17.27
2	7060	1.211626	-	-	17.26
2	7080	1.241229	-	-	17.26
2	7100	1.212566	-	-	17.31

```
   2    |  7120   |   1.242812   |     -      |     -     |   17.30
   2    |  7140   |   1.182900   |     -      |     -     |   17.30
   2    |  7160   |   1.307001   |     -      |     -     |   17.25
   2    |  7180   |   1.196482   |     -      |     -     |   17.33
   2    |  7200   |   1.209605   |     -      |     -     |   17.25
   2    |  7220   |   1.306756   |     -      |     -     |   17.31
   2    |  7240   |   1.115354   |     -      |     -     |   17.26
   2    |  7260   |   1.310647   |     -      |     -     |   17.30
   2    |  7280   |   1.222305   |     -      |     -     |   17.27
   2    |  7300   |   1.242259   |     -      |     -     |   17.26
   2    |  7320   |   1.210017   |     -      |     -     |   17.32
   2    |  7340   |   1.233105   |     -      |     -     |   17.27
   2    |  7360   |   1.231795   |     -      |     -     |   17.27
   2    |  7380   |   1.263031   |     -      |     -     |   17.25
   2    |  7400   |   1.213477   |     -      |     -     |   17.23
   2    |  7420   |   1.243056   |     -      |     -     |   17.32
   2    |  7440   |   1.230327   |     -      |     -     |   17.25
   2    |  7460   |   1.199413   |     -      |     -     |   17.31
   2    |  7480   |   1.011393   |     -      |     -     |   17.30
   2    |  7500   |   1.259258   |     -      |     -     |   17.27
   2    |  7520   |   1.283490   |     -      |     -     |   17.28
   2    |  7540   |   1.288913   |     -      |     -     |   17.24
   2    |  7560   |   1.321083   |     -      |     -     |   17.33
   2    |  7580   |   1.302783   |     -      |     -     |   17.24
   2    |  7600   |   1.253574   |     -      |     -     |   17.32
   2    |  7620   |   1.165328   |     -      |     -     |   17.27
   2    |  7640   |   1.233758   |     -      |     -     |   17.31
   2    |  7660   |   1.060169   |     -      |     -     |   17.25
   2    |  7680   |   1.197774   |     -      |     -     |   17.28
   2    |  7700   |   1.174415   |     -      |     -     |   17.30
   2    |  7720   |   1.254955   |     -      |     -     |   17.28
   2    |  7740   |   1.151981   |     -      |     -     |   17.29
   2    |  7760   |   1.267728   |     -      |     -     |   17.28
   2    |  7780   |   1.126012   |     -      |     -     |   17.26
   2    |  7800   |   1.150836   |     -      |     -     |   17.27
   2    |  7820   |   1.165800   |     -      |     -     |   17.28
   2    |  7840   |   1.231077   |     -      |     -     |   17.37
   2    |  7860   |   1.350720   |     -      |     -     |   17.30
   2    |  7874   |   1.232936   |     -      |     -     |   12.13
---------------------------------------------------------------------
   2    |    -    |   1.289786   |  1.229674  |   63.41   |  7358.45
---------------------------------------------------------------------

Training complete!
```

+ Code + Markdown

Saving & loading the model

```
torch.save(bert_classifier, 'model_DL.pt')
model_DL= torch.load('model_DL.pt')
```

Predict and evaluate validation subset

+ Code + Markdown

[49]:
```
probs_bert_classifier = bert_predict(bert_classifier, val_dataloader)
evaluate_roc(probs_bert_classifier, y_val)
```

```
              precision    recall  f1-score   support

          AE       0.74      0.54      0.62      7500
          BH       0.91      0.47      0.62      7500
          DZ       0.73      0.72      0.72      7500
          EG       0.58      0.78      0.67      7500
          IQ       0.66      0.73      0.69      7500
          JO       0.79      0.49      0.60      7500
          KW       0.49      0.56      0.52      7500
          LB       0.60      0.75      0.66      7500
          LY       0.70      0.63      0.66      7500
          MA       0.74      0.81      0.77      7500
          OM       0.38      0.66      0.48      7500
          PL       0.70      0.47      0.56      7500
          QA       0.66      0.66      0.66      7500
          SA       0.39      0.57      0.46      7500
          SD       0.75      0.73      0.74      7500
          SY       0.76      0.56      0.64      7500
          TN       0.84      0.73      0.78      7500
          YE       0.63      0.56      0.59      7500

    accuracy                           0.63    135000
   macro avg       0.67      0.63      0.64    135000
weighted avg       0.67      0.63      0.64    135000
```

+ Code + Markdown

Predict and evaluate Test subset

```
# Run `preprocessing_for_bert` on the test set
test_inputs, test_masks = preprocessing_for_bert(x_test)
# x_test,  y_test
# Create the DataLoader for our test set
test_dataset = TensorDataset(test_inputs, test_masks)
test_sampler = SequentialSampler(test_dataset)
test_dataloader = DataLoader(test_dataset, sampler=test_sampler, batch_size=32)
```

[49]:
```
# Compute predicted probabilities on the test set
probs = bert_predict(bert_classifier, test_dataloader)
evaluate_roc(probs, y_test)
```

	precision	recall	f1-score	support
AE	0.74	0.54	0.62	7500
BH	0.91	0.47	0.62	7500
DZ	0.73	0.72	0.72	7500
EG	0.58	0.78	0.67	7500
IQ	0.66	0.73	0.69	7500
JO	0.79	0.49	0.60	7500
KW	0.49	0.56	0.52	7500
LB	0.60	0.75	0.66	7500
LY	0.70	0.63	0.66	7500
MA	0.74	0.81	0.77	7500
OM	0.38	0.66	0.48	7500
PL	0.70	0.47	0.56	7500
QA	0.66	0.66	0.66	7500
SA	0.39	0.57	0.46	7500
SD	0.75	0.73	0.74	7500
SY	0.76	0.56	0.64	7500
TN	0.84	0.73	0.78	7500
YE	0.63	0.56	0.59	7500
accuracy			0.63	135000
macro avg	0.67	0.63	0.64	135000
weighted avg	0.67	0.63	0.64	135000

+ Code + Markdown

Summary of performance on test datasets

Model	Accuracy (%)
RandomForestClassifier	51
Multi-dialect-Arabic-BERT	64

[ ]:

Console