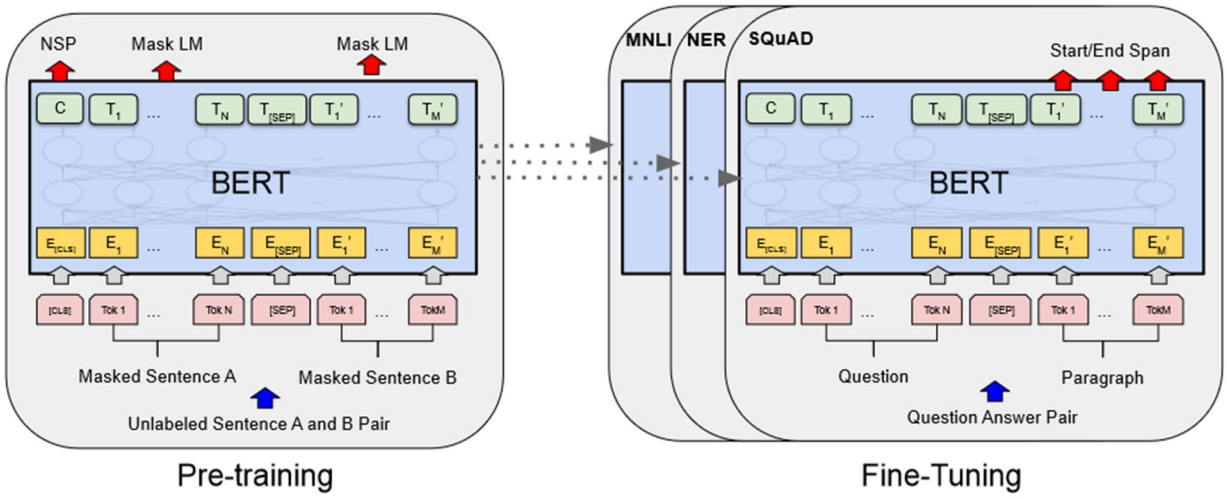
Natural Language Processing Course

CA#4 (Transformers & Bert) Report

Spring 1401

Parnian Fazel – 810198516

Bidirectional **E**ncoder **R**epresentations from **T**ransformers



Contents

Question 1: ParsiNLU dataset Classification

In this question, we intend to use Transformers in text application. One of the applications in the Natural Language Processing studied is Textual entailment. Textual entailment is a classification that can be used to understand the relationship between two sentences. In this assignment there are 3 relationships:

- ✓ **Entailment:** In this case, the first sentence proves that the second sentence is correct.
- ✓ **Contradiction:** There is contradiction between the two sentences.
- ✓ **Neutral:** The sentences are not related to each other.

In this task we are to use Contextualized word embedding like BERT.
For the purpose of classification, I created a class:

I this question I created a class named **Classifer**:

```
class Classifier:
    def __init__(self, model_checkpoint, train_column_name, test_column_name, preprocess = False, batch_size=32, seq_len=128):
      self.dataset = load_dataset("persiannlp/parsinlu_entailment")
      self.dataset = self.dataset.filter(lambda record: record['label'] in ["c", "e", "n"])
      self.encoded_dict = {'c' : 0, 'e': 1, 'n': 2}
      self.seq_len = seq_len
      self.model_checkpoint = model_checkpoint
      self.train_column_name = train_column_name
      self.test_column_name = test_column_name
      self.batch_size = batch_size
      self.preprocess = preprocess
      self.prepare_data()
      self.set_model_and_tokenizer()

    def encode_classes(self, df):
      return df.label.map(self.encoded_dict)

    def prepare_data(self):
      self.train_df = pd.DataFrame(self.dataset['train'])
      self.valid_df = pd.DataFrame(self.dataset['validation'])
      self.test_df = pd.DataFrame(self.dataset['test'])
      self.train_df['label'] = self.encode_classes(self.train_df)
      self.valid_df['label'] = self.encode_classes(self.valid_df)
      self.test_df['label'] = self.encode_classes(self.test_df)
      if self.preprocess:
        self.train_df = self.clean_data(self.train_df)
        self.valid_df = self.clean_data(self.valid_df)
        self.test_df = self.clean_data(self.test_df)

    def set_model_and_tokenizer(self):
      if self.model_checkpoint != 'xlm-roberta-base':
        self.tokenizer = AutoTokenizer.from_pretrained(self.model_checkpoint)
        self.bert = TFAutoModel.from_pretrained(self.model_checkpoint)
      else:
        self.tokenizer = AutoTokenizer.from_pretrained(self.model_checkpoint, from_pt=True)
        self.bert = TFAutoModel.from_pretrained(self.model_checkpoint, from_pt=True)
      self.bert.trainable = True
```

```
def set_model_and_tokenizer(self):
  if self.model_checkpoint != 'xlm-roberta-base':
    self.tokenizer = AutoTokenizer.from_pretrained(self.model_checkpoint)
    self.bert = TFAutoModel.from_pretrained(self.model_checkpoint)
  else:
    self.tokenizer = AutoTokenizer.from_pretrained(self.model_checkpoint, from_pt=True)
    self.bert = TFAutoModel.from_pretrained(self.model_checkpoint, from_pt=True)
  self.bert.trainable = True

def get_tokenized_ids(self, df, column_name):
  record_len = len(df)
  input_ids = np.zeros((record_len, self.seq_len))
  mask_ids = np.zeros((record_len, self.seq_len))
  for i, row in df.iterrows():
    input_data = row['sent1'], row['sent2']
    tokens = self.tokenizer.encode_plus(input_data, max_length=self.seq_len, truncation=True, padding='max_length', add_special_tokens=True, return_tensors='tf')
    input_ids[i, :] = tokens['input_ids']
    mask_ids[i, :] = tokens['attention_mask']
  class_values = df['label'].values
  labels = np.zeros((record_len, class_values.max()+1))
  labels[np.arange(record_len), class_values] = 1 #one-hot encoding
  return input_ids, mask_ids, labels

def dataset_mapper(self, input_ids, masks, labels):
  return {'input_ids': input_ids, 'attention_mask': masks}, labels

def get_dataset(self, df, column_name):
  input_ids, mask_ids, labels = self.get_tokenized_ids(df, column_name)
  dataset = tf.data.Dataset.from_tensor_slices((input_ids, mask_ids, labels))
  dataset = dataset.map(self.dataset_mapper)
  return dataset.batch(self.batch_size, drop_remainder=True)
```

```
  def create_model(self):
    input_ids = tf.keras.layers.Input(shape=(128,), name='input_ids', dtype='int32')
    mask = tf.keras.layers.Input(shape=(128,), name='attention_mask', dtype='int32')
    embeddings = self.bert(input_ids, attention_mask=mask).pooler_output
    drop_output = tf.keras.layers.Dropout(0.6)(embeddings)
    output = tf.keras.layers.Dense(1024, activation='relu')(drop_output)
    output = tf.keras.layers.Dense(3, activation='softmax', name='outputs')(output)
    model = tf.keras.Model(inputs=[input_ids, mask], outputs=output)
    optimizer = tf.keras.optimizers.Adam(learning_rate=1e-5)
    loss = tf.keras.losses.CategoricalCrossentropy()
    acc = tf.keras.metrics.CategoricalAccuracy('accuracy')
    model.compile(optimizer=optimizer, loss=loss, metrics=[acc])
    print(model.summary())
    return model

  def train_and_evaluate_model(self, model):
    train_dataset = self.get_dataset(self.train_df, self.train_column_name)
    valid_dataset = self.get_dataset(self.valid_df, self.train_column_name)
    input_ids_test, mask_ids_test, labels_test = self.get_tokenized_ids(self.test_df, self.test_column_name)
    history = model.fit(train_dataset, validation_data=valid_dataset, epochs=10)
    score = model.evaluate([input_ids_test, mask_ids_test], labels_test, verbose=0)
    predictions = np.argmax(model.predict([input_ids_test, mask_ids_test]), axis=1)
    self.print_results(score, predictions, self.test_df['label'])
    self.plot_results(history)
    self.print_confusion_matrix(self.test_df['label'].to_list(), predictions)
    return model, history, score, predictions

  def clean_text(self, text):
    lemmatizer = WordNetLemmatizer()
    tokens = nltk.word_tokenize(text)
    tokens = [word.lower() for word in tokens]
    tokens = [word for word in tokens if not word in nltk_stopwords.words("english")]
    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    return tokens

  def clean_data(self, df):
    for i, row in df.iterrows():
      row['sent1'] =  self.clean_text(row['sent1'])
      row['sent2'] =  self.clean_text(row['sent2'])
    return df
```

Part 1

After loading the dataset from "persiannlp/parsinlu_entailment", we can see the dataset:

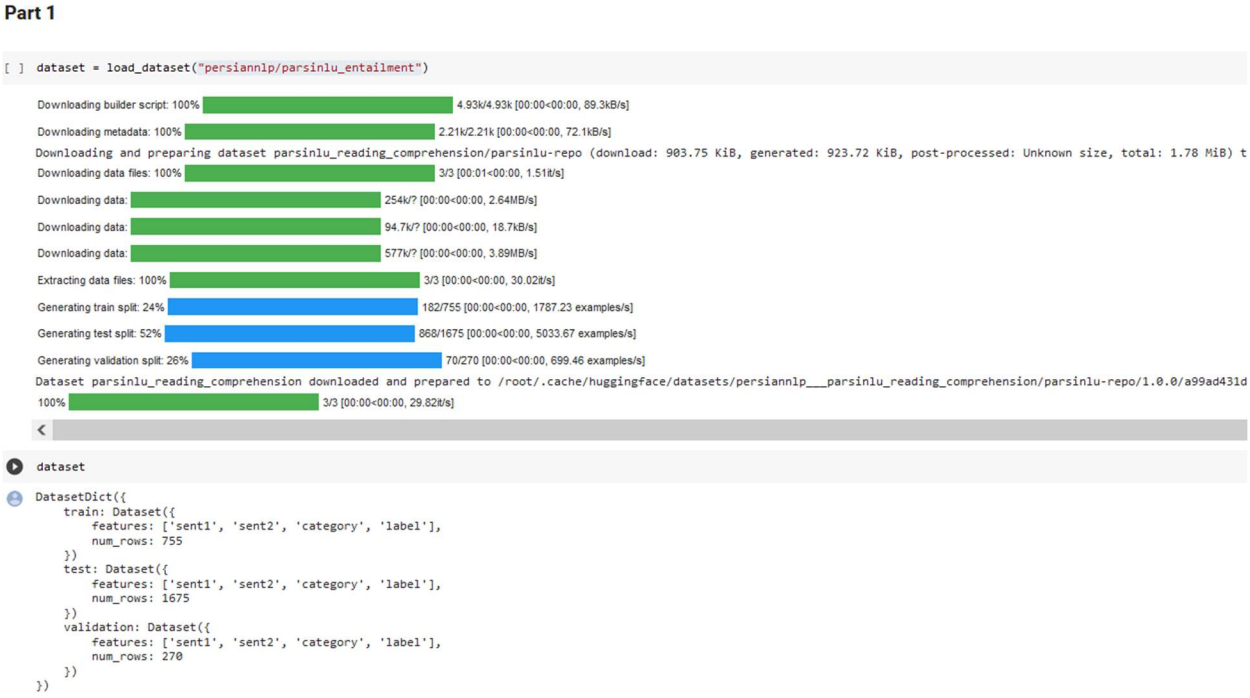


Figure 1-Given dataset of question 1

```
train_df = pd.DataFrame(dataset['train'])
test_df = pd.DataFrame(dataset['test'])
train_df
```

	sent1	sent2	category	label
0	...زبان به قدری بخش بزرگی از نیروی کار را تشکیل م	...مردان بخش عظیمی از نیروی کار هستند بنابراین تن	translation-train	c
1	...سالها است که کنگره در تلاش است تا اثربخشی مدیر	...کنگره بودجه ویژه ای برای مدیریت اطلاعات و فناو	translation-train	n
2	...سرامیک‌های زیست خنثی پس از قرارگیری در بدن میز	.خواص فیزیکی سرامیک‌ها قابل اندازه گیری است	natural-wiki	n
3	...دولت از هیچ قانونی که منجر به کاهش چشمگیر توان	...قانونی که باعث کاهش استفاده از زغال سنگ به علو	translation-train	e
4	...روش‌ها و الگوریتم‌های بهینه‌سازی به دو دسته ال	.آمار در دروس مدیریتی نقش مهمی را بازی می‌کند	natural-wiki	n
...	...	...	...	...
750	...قاسم رمضانپور پس از فارغ‌التحصیلی در مقطع کارش	...قاسم رمضان پور در آزمون اعزام به خارج، پذیرفته	natural-wiki	c
751	...اما پرزیدنت بوش همیشه احساس کرده است که سازمان	.پرزیدنت بوش بر نقش مهم سازمان ملل متحد تاکید کرد	natural-voa	e
752	...ویلاهای زیادی در تمام مسیر قرار دارد ولی کم کم	... ویلاهای بزرگ زیادی در طول مسیر وجود دارند ولی	translation-dev	n
753	LA خوب من فکر می کنم قانون	...است زیرا یکی از دوس LA من فکر می کنم این قانون	translation-train	n
754	.من فکر نمی کنم آنها حس مسئولیت داشته باشند	...من فکر نمی کنم آنها احساس مسئولیت کنند که دیگر	translation-train	n

755 rows × 4 columns

Figure 2-Training Data of question 1

```
print(train_df['label'].value_counts())
plt.title("Train dataset value counts")
train_df['label'].value_counts().plot.bar(color='maroon')
```

```
e     304
n     245
c     205
xx      1
Name: label, dtype: int64
<matplotlib.axes._subplots.AxesSubplot at 0x7f19cc924c10>
```

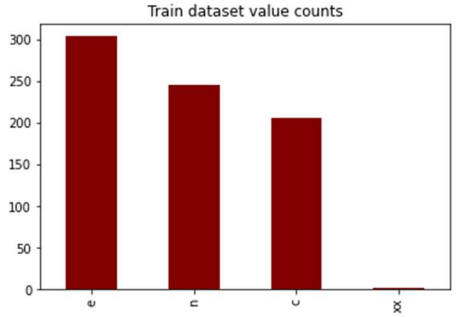


Figure 3-Train data value counts of question 1 classes

```
[ ] print(test_df['label'].value_counts())
    plt.title("Test dataset value counts")
    test_df['label'].value_counts().plot.bar(color='maroon')
```

```
e    610
c    561
n    502
-      2
Name: label, dtype: int64
<matplotlib.axes._subplots.AxesSubplot at 0x7f19cc85c250>
```

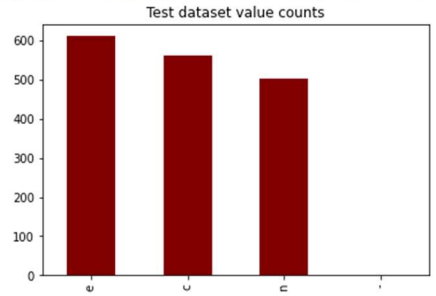


Figure 4-Test data value counts of question 1 classes

In this question it is not necessary to pre-process data. The contextual word embedding models which we are going to use in this task are xlm-roberta and parsbert, which use BPE for tokenization and are generalized so it will make a good embedding for our dataset, hence we do not need to pre-process data. Also, pay attention that the given dataset is clean itself.
First, I thought as these models almost consider everything, if we for instance use lemmatization or stemming, we kind of ruin the wordform of it and then even the Bert model can not produce a good embedding for it. Imagine the word 'computable' will be "compute" + "able" in Bert, on the other hand, by stemming we will get "comput" which the Bert model will not understand this as it was 'computable'! Also, I imagined in Textual entailment the punctuations and a lot of stop-words are effective in classification. Imagine two sentences have a contradiction relation; these sentences should have the same form. I mean one of them can not be a statement and the other be a question. Therefore, removing stop-words and punctuations can lead to decrease performance. But when I applied preprocessing, I got slightly better results! Although to be sure about the results we should run the model several times and consider the range of results we get from this model. I will show both preprocessed and not preprocessed results of part 1 and part 2. I can mention that I dropped a few records of dataset which did not have a valid label.

Part 2

There are details of xlm-roberta model shown:

XLM-RoBERTa	`xlm-roberta-base`	~125M parameters with 12-layers, 768-hidden-state, 3072 feed-forward hidden-state, 8-heads, Trained on on 2.5 TB of newly created clean CommonCrawl data in 100 languages
	`xlm-roberta-large`	~355M parameters with 24-layers, 1027-hidden-state, 4096 feed-forward hidden-state, 16-heads, Trained on 2.5 TB of newly created clean CommonCrawl data in 100 languages

Figure 5-XLM-RoBERTa details of implementation

I used xlm-roberta-base.
In this part I created a neural network with a word-embedding layer of Multiningual BERT, XLM-RoBERTa which is available here. For Fine-tuning I used the **attention mask ids** and, also the **trainable attribute is True**. In my network there is a hidden layer with 1024 cells and I added a dropout layer to prevent overfitting. First, I encoded the 'label' column with 0, 1 and 2. Then I used AutoTokenizer.from_pretrained() from Transformers library and set the checkpoint to 'xlm-roberta-base'. Then, I used **encode_plus** to tokenize texts. This function gives us the input ids and mask ids, and also create a "[**CLS**] + sentence1 + [**SEP**] + sentence2". This [SEP] indicates the separation of two sentences. The attention mask ids are to prevent the model from looking at **padding** tokens. I created a dataset by these ids and the labels. I passed these datasets to fit the model and predict the test dataset.

Part 2

```
[ ] classifier = Classifier('xlm-roberta-base', '', '',  False, batch_size=8, seq_len=128)
    part2_model = classifier.create_model()
    part2_model, part2_history, part2_score, part2_predictions = classifier.train_and_evaluate_model(part2_model)
```

Figure 6-Running model in python

The summary of model:

```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
input_ids (InputLayer)       [(None, 128)]             0           []

attention_mask (InputLayer)  [(None, 128)]             0           []

tfxlm_roberta_model_2 (TFXLMRo  TFBaseModelOutputWi  278043648  ['input_ids[0][0]',
bertaModel)                  thPoolingAndCrossAt                    'attention_mask[0][0]']
                             tentions(last_hidde
                             n_state=(None, 128,
                              768),
                              pooler_output=(Non
                             e, 768),
                              past_key_values=No
                             ne, hidden_states=N
                             one, attentions=Non
                             e, cross_attentions
                             =None)

dropout_113 (Dropout)        (None, 768)               0           ['tfxlm_roberta_model_2[0][1]']

dense_4 (Dense)              (None, 1024)              787456      ['dropout_113[0][0]']

outputs (Dense)              (None, 3)                 3075        ['dense_4[0][0]']

==================================================================================================
Total params: 278,834,179
Trainable params: 278,834,179
Non-trainable params: 0
```

Figure 7-Summary of the model in question 1 part 2

I trained the model by 10 epoches:

```
Epoch 1/10
94/94 [==============================] - 41s 350ms/step - loss: 1.1244 - accuracy: 0.3537 - val_loss: 1.1432 - val_accuracy: 0.2841
Epoch 2/10
94/94 [==============================] - 31s 328ms/step - loss: 1.1172 - accuracy: 0.3750 - val_loss: 1.1499 - val_accuracy: 0.2841
Epoch 3/10
94/94 [==============================] - 32s 337ms/step - loss: 1.1165 - accuracy: 0.3670 - val_loss: 1.1535 - val_accuracy: 0.2841
Epoch 4/10
94/94 [==============================] - 31s 329ms/step - loss: 1.1169 - accuracy: 0.3803 - val_loss: 1.1539 - val_accuracy: 0.2841
Epoch 5/10
94/94 [==============================] - 31s 329ms/step - loss: 1.1103 - accuracy: 0.3723 - val_loss: 1.1396 - val_accuracy: 0.2841
Epoch 6/10
94/94 [==============================] - 31s 332ms/step - loss: 1.0978 - accuracy: 0.3989 - val_loss: 1.1570 - val_accuracy: 0.2841
Epoch 7/10
94/94 [==============================] - 31s 327ms/step - loss: 1.1081 - accuracy: 0.3511 - val_loss: 1.1449 - val_accuracy: 0.2841
Epoch 8/10
94/94 [==============================] - 31s 332ms/step - loss: 1.1081 - accuracy: 0.3737 - val_loss: 1.1324 - val_accuracy: 0.2841
Epoch 9/10
94/94 [==============================] - 31s 330ms/step - loss: 1.0862 - accuracy: 0.4056 - val_loss: 1.1277 - val_accuracy: 0.3106
Epoch 10/10
94/94 [==============================] - 31s 329ms/step - loss: 1.0441 - accuracy: 0.4535 - val loss: 1.1134 - val accuracy: 0.3826
```

Figure 8-Result of 10 epoches after fitting the model in question 1 part 2

Also, the wanted results (**without preprocessing**) are:

```
Loss on test data: 1.081238865852356
Accuracy on test data: 0.4476987421512604
Classification Report:
              precision    recall  f1-score   support

           c       0.00      0.00      0.00       561
           e       0.45      0.70      0.55       610
           n       0.44      0.64      0.52       502

    accuracy                           0.45      1673
   macro avg       0.30      0.45      0.36      1673
weighted avg       0.30      0.45      0.36      1673

AUC - Contradiction:  0.47794678695546244
AUC - Entailment:  0.3907206637570747
AUC - Neutral:  0.6439459922904454
```

Figure 9-Result & classification report of the model without pre-processing in question 1 part 2

As we can see the result is **45%**.

The wanted results (**with preprocessing**) are:

```
Epoch 9/10
94/94 [==============================] - 31s 329ms/step - loss: 0.9241 - accuracy: 0.5691 - val_loss: 1.0769 - val_accuracy: 0.4470
Epoch 10/10
94/94 [==============================] - 31s 332ms/step - loss: 0.7565 - accuracy: 0.6875 - val_loss: 1.2823 - val_accuracy: 0.4621
Loss on test data: 1.25709068775177
Accuracy on test data: 0.49193066358566284
Classification Report:
              precision    recall  f1-score   support

           c       0.62      0.29      0.39       561
           e       0.50      0.51      0.51       610
           n       0.44      0.70      0.54       502

    accuracy                           0.49      1673
   macro avg       0.52      0.50      0.48      1673
weighted avg       0.52      0.49      0.48      1673

AUC - Contradiction:  0.37728667333512866
AUC - Entailment:  0.4712436192033064
AUC - Neutral:  0.6619465774817042
```

Figure 10-Result & classification report of the model with pre-processing in question 1 part 2

As we can see the result is **49%**.

I plotted some results in order to study the performance of the model:

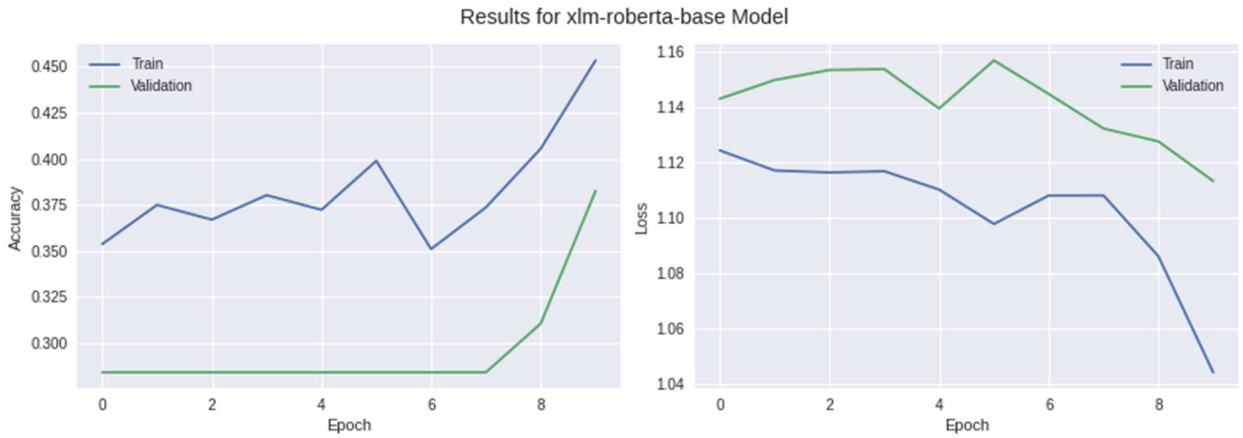


Figure 11-Accuracy & Loss plots in question 1 part 2

The confusion matrix can be shown here:

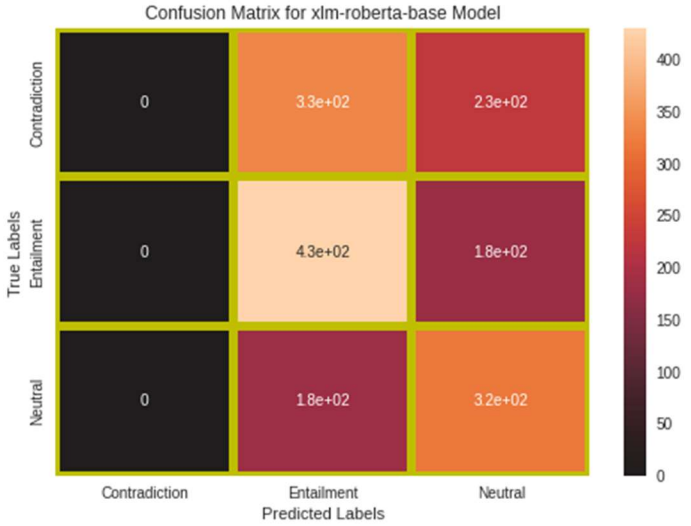


Figure 12-Confusion Matrix in question 1 part 2

Recent results of different papers show that **multilingual** models show great performance even when evaluated on a single language. XLM-RoBERTa is trained on a multilingual language modeling **objective** using only monolingual data. This basically means samples of text streams are taken from all the languages, and the model is trained to predict masked tokens in the input. XLM-RoBERTa is a multilingual model trained on 100 different languages. Unlike some XLM multilingual models, it does not require lang tensors to understand which language is used and should be able to determine the correct language from the **input ids**.

Part 3

In this part I used the created a neural network in the last part with a word-embedding layer of ParsBERT which is available here. For **Fine-tuning** I used the **attention mask ids** and, also the **trainable attribute is True**. I can mention that the size of xlm-roberta vocabulary is lot more than ParsBert so it is more complex and slower.
The parameters and the procedure of creating the model is same as part 2.

Part 3

```
[ ] classifier = Classifier('HooshvareLab/bert-base-parsbert-uncased', '', '',  batch_size=8, seq_len=128)
    part3_model = classifier.create_model()
    part3_model, part3_history, part3_score, part3_predictions = classifier.train_and_evaluate_model(part3_model)
```

Figure 13-Running the model in python

The summary of model:

Layer (type)	Output Shape	Param #	Connected to
input_ids (InputLayer)	[(None, 128)]	0	[]
attention_mask (InputLayer)	[(None, 128)]	0	[]
tf_bert_model (TFBertModel)	TFBaseModelOutputWithPoolingAndCrossAttentions(last_hidden_state=(None, 128, 768), pooler_output=(None, 768), past_key_values=None, hidden_states=None, attentions=None, cross_attentions=None)	162841344	['input_ids[0][0]', 'attention_mask[0][0]']
dropout_151 (Dropout)	(None, 768)	0	['tf_bert_model[0][1]']
dense_5 (Dense)	(None, 1024)	787456	['dropout_151[0][0]']
outputs (Dense)	(None, 3)	3075	['dense_5[0][0]']

```
Total params: 163,631,875
Trainable params: 163,631,875
Non-trainable params: 0
```

Figure 14-Summary of the model in question 1 part 3

I trained the model by 10 epoches:

```
Epoch 1/10
94/94 [==============================] - 36s 295ms/step - loss: 1.1859 - accuracy: 0.3630 - val_loss: 1.1169 - val_accuracy: 0.3295
Epoch 2/10
94/94 [==============================] - 26s 278ms/step - loss: 1.1179 - accuracy: 0.4162 - val_loss: 1.1028 - val_accuracy: 0.3788
Epoch 3/10
94/94 [==============================] - 26s 274ms/step - loss: 1.0081 - accuracy: 0.5053 - val_loss: 1.1478 - val_accuracy: 0.3864
Epoch 4/10
94/94 [==============================] - 26s 276ms/step - loss: 0.8756 - accuracy: 0.6077 - val_loss: 1.1204 - val_accuracy: 0.4205
Epoch 5/10
94/94 [==============================] - 26s 273ms/step - loss: 0.6678 - accuracy: 0.7460 - val_loss: 1.2034 - val_accuracy: 0.4811
Epoch 6/10
94/94 [==============================] - 26s 277ms/step - loss: 0.4024 - accuracy: 0.8590 - val_loss: 1.6424 - val_accuracy: 0.4394
Epoch 7/10
94/94 [==============================] - 26s 274ms/step - loss: 0.2001 - accuracy: 0.9468 - val_loss: 1.6012 - val_accuracy: 0.4735
Epoch 8/10
94/94 [==============================] - 26s 274ms/step - loss: 0.1362 - accuracy: 0.9601 - val_loss: 1.8850 - val_accuracy: 0.4621
Epoch 9/10
94/94 [==============================] - 26s 275ms/step - loss: 0.0537 - accuracy: 0.9934 - val_loss: 2.7670 - val_accuracy: 0.4318
Epoch 10/10
94/94 [==============================] - 26s 276ms/step - loss: 0.0468 - accuracy: 0.9880 - val_loss: 2.5630 - val_accuracy: 0.4432
```

Figure 15-Result of 10 epoches after fitting the model in question 1 part 3

Also, the wanted results (**without preprocessing**) are:

```
Loss on test data: 2.4083404541015625
Accuracy on test data: 0.479976087808609
Classification Report:
              precision    recall  f1-score   support

           c       0.50      0.27      0.35       561
           e       0.47      0.71      0.56       610
           n       0.50      0.44      0.46       502

    accuracy                           0.48      1673
   macro avg       0.49      0.47      0.46      1673
weighted avg       0.49      0.48      0.46      1673

AUC - Contradiction:  0.4196241936931738
AUC - Entailment:  0.47064293755686815
AUC - Neutral:  0.6176795805675674
```

Figure 16-Result & classification report of the model without pre-processing in question 1 part 3

As we can see the result is **48%**.

The wanted results (**with preprocessing**) are:

```
Epoch 9/10
94/94 [==============================] - 26s 273ms/step - loss: 0.1551 - accuracy: 0.9535 - val_loss: 1.7433 - val_accuracy: 0.4773
Epoch 10/10
94/94 [==============================] - 26s 277ms/step - loss: 0.0800 - accuracy: 0.9814 - val_loss: 2.0113 - val_accuracy: 0.4886
Loss on test data: 1.9647579193115234
Accuracy on test data: 0.5044829845428467
Classification Report:
              precision    recall  f1-score   support

           c       0.50      0.40      0.44       561
           e       0.49      0.64      0.56       610
           n       0.53      0.45      0.49       502

    accuracy                           0.50      1673
   macro avg       0.51      0.50      0.50      1673
weighted avg       0.51      0.50      0.50      1673

AUC - Contradiction:  0.37891211095294886
AUC - Entailment:  0.499539657326157
AUC - Neutral:  0.6290091555213817
```

Figure 17-Result & classification report of the model with pre-processing in question 1 part 3

As we can see the result is **50%**.

I plotted some results in order to study the performance of the model:

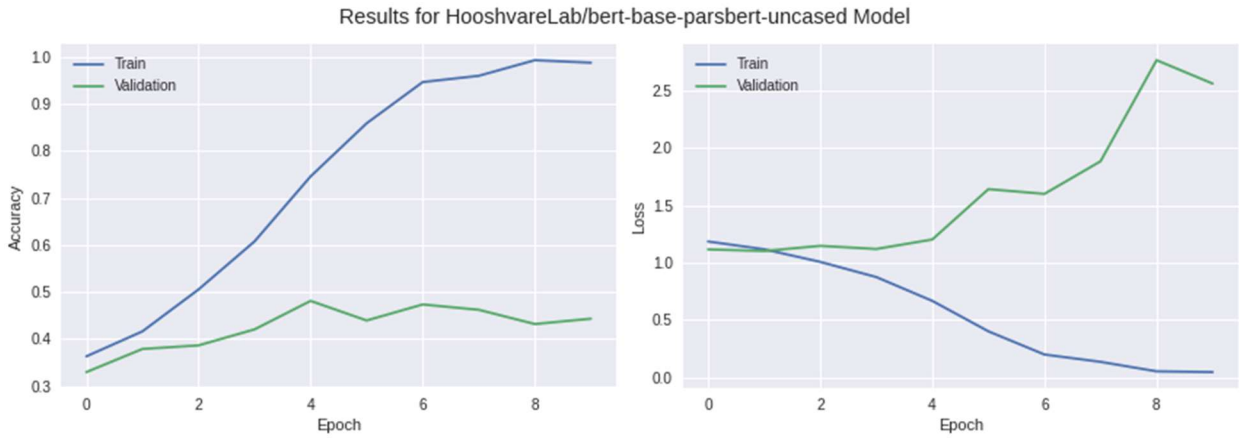


Figure 18-Accuracy & Loss plots in question 1 part 3

The confusion matrix can be shown here:

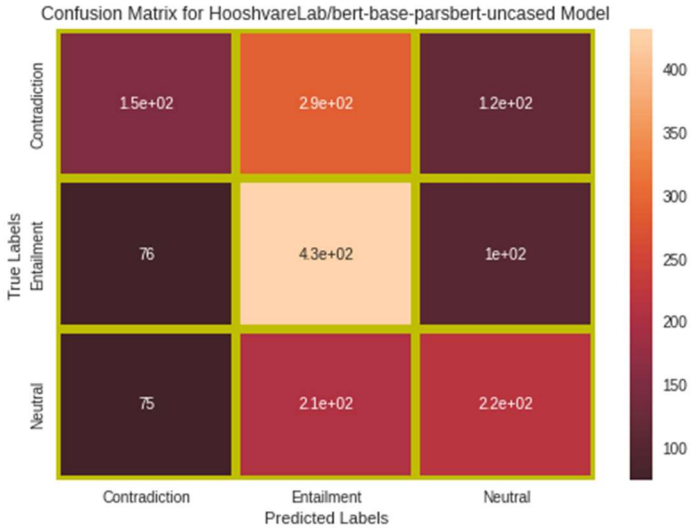


Figure 19-Confusion Matrix in question 1 part 3

The result of part 2 and part 3 shows that the ParsiBert model has slight better performance in this task. This is expectable since the dataset is Persian. The ParsiBert model weights and all the parameters is designed especially for Persian language but the xlm-roberta model is for 100 languages. However, both models got somehow same results.

Question 2: Multilingual classification

Architecture	Shortcut name	Details of the model
BERT	`bert-base-uncased`	12-layer, 768-hidden, 12-heads, 110M parameters. Trained on lower-cased English text.
	`bert-large-uncased`	24-layer, 1024-hidden, 16-heads, 340M parameters. Trained on lower-cased English text.
	`bert-base-cased`	12-layer, 768-hidden, 12-heads, 110M parameters. Trained on cased English text.
	`bert-large-cased`	24-layer, 1024-hidden, 16-heads, 340M parameters. Trained on cased English text.

Figure 20-Bert details of implementation

In this question I created a class named Classifier which pretty much the same as the class in question 1:

```
class Classifier:
    def __init__(self, model_address, train_column_name, test_column_name, batch_size=32, seq_len=128):
      self.train_df = pd.read_excel('/content/train.xlsx')
      self.valid_df = pd.read_excel('/content/valid.xlsx')
      self.test_df = pd.read_excel('/content/test.xlsx')
      self.encoded_dict = {'quran' : 0, 'bible': 1, 'mizan': 2}
      self.seq_len = seq_len
      self.model_address = model_address
      self.train_column_name = train_column_name
      self.test_column_name = test_column_name
      self.batch_size = batch_size
      self.prepare_data()
      self.set_model_and_tokenizer()

    def encode_classes(self, df):
      return df.category.map(self.encoded_dict)

    def prepare_data(self):
      self.train_df['category'] = self.encode_classes(self.train_df)
      self.valid_df['category'] = self.encode_classes(self.valid_df)
      self.test_df['category'] = self.encode_classes(self.test_df)

    def set_model_and_tokenizer(self):
      if self.model_address != 'xlm-roberta-base':
        self.tokenizer = AutoTokenizer.from_pretrained(self.model_address)
        self.bert = TFAutoModel.from_pretrained(self.model_address)
      else:
        self.tokenizer = AutoTokenizer.from_pretrained(self.model_address, from_pt=True)
        self.bert = TFAutoModel.from_pretrained(self.model_address, from_pt=True)
```

```
  def get_tokenized_ids(self, df, column_name):
    record_len = len(df)
    Xinput_ids = np.zeros((record_len, self.seq_len))
    Xmask_ids = np.zeros((record_len, self.seq_len))

    for i, row in df.iterrows():
      if self.model_address != 'xlm-roberta-base':
        input_data = row[column_name]
      else:
        input_data = row['source'], row['targets']

      tokens = self.tokenizer.encode_plus(input_data, max_length=self.seq_len, truncation=True,
                                          padding='max_length', add_special_tokens=True,
                                          return_tensors='tf')
      Xinput_ids[i, :] = tokens['input_ids']
      Xmask_ids[i, :] = tokens['attention_mask']

    class_values = df['category'].values
    labels = np.zeros((record_len, class_values.max()+1))
    labels[np.arange(record_len), class_values] = 1 #one-hot encoding
    return Xinput_ids, Xmask_ids, labels


  def dataset_mapper(self, input_ids, masks, labels):
    return {'input_ids': input_ids, 'attention_mask': masks}, labels
```

```
def get_dataset(self, df, column_name):
  Xinput_ids, Xmask_ids, labels = self.get_tokenized_ids(df, column_name)
  dataset = tf.data.Dataset.from_tensor_slices((Xinput_ids, Xmask_ids, labels))
  dataset = dataset.map(self.dataset_mapper)
  return dataset.batch(self.batch_size, drop_remainder=True)
```

```
def create_model(self):
  input_ids = tf.keras.layers.Input(shape=(128,), name='input_ids', dtype='int32')
  mask = tf.keras.layers.Input(shape=(128,), name='attention_mask', dtype='int32')

  embeddings = self.bert(input_ids, attention_mask=mask).pooler_output
  drop_output = tf.keras.layers.Dropout(0.7)(embeddings)
  output = tf.keras.layers.Dense(1024, activation='relu')(drop_output)
  y = tf.keras.layers.Dense(3, activation='softmax', name='outputs')(output)

  model = tf.keras.Model(inputs=[input_ids, mask], outputs=y)
  #model.layers[2].trainable = False #for freezing bert layer
  optimizer = tf.keras.optimizers.Adam(learning_rate=3e-5)
  loss = tf.keras.losses.CategoricalCrossentropy()
  acc = tf.keras.metrics.CategoricalAccuracy('accuracy')
  model.compile(optimizer=optimizer, loss=loss, metrics=[acc])
  print(model.summary())
  return model

def train_and_evaluate_model(self, model):
  train_dataset = self.get_dataset(self.train_df, self.train_column_name)
  valid_dataset = self.get_dataset(self.valid_df, self.train_column_name)
  Xinput_ids_test, Xmask_ids_test, labels_test = self.get_tokenized_ids(self.test_df, self.test_column_name)
  history = model.fit(train_dataset, validation_data=valid_dataset, epochs=10)
  score = model.evaluate([Xinput_ids_test, Xmask_ids_test], labels_test, verbose=0)
  print("Loss on test data:", score[0])
  print("Accuracy on test data:", score[1])
  print("Classification Report:")
  predictions = np.argmax(model.predict([Xinput_ids_test, Xmask_ids_test]), axis=1)
  print(classification_report(self.test_df['category'], predictions, target_names=['quran', 'bible', 'mizan']))

  for i, classification in enumerate(["quran", "bible", "mizan"]):
    fpr, tpr, thresholds = metrics.roc_curve(np.array(self.test_df['category'].to_list()), np.array(predictions), pos_label=i)
    auc_res = metrics.auc(fpr, tpr)
    print(f"AUC - {classification}: ", auc_res)

  self.plot_results(history)
  self.print_confusion_matrix(self.test_df['category'].to_list(), predictions)
  return model, history, score, predictions
```

It is useful to take look at given data:

```
train_df = pd.read_excel('/content/train.xlsx')
test_df = pd.read_excel('/content/test.xlsx')
train_df
```

	source	targets	category
0	When news is brought to one of them, of (the b...	...و چون یکی از آنان را به [ولادت] دختر مژده دهند	quran
1	After them repaired Zadok the son of Immer ove...	...و چون دشمنان ما شنیدند که ما آگاه شده‌ایم و خد	bible
2	And establish regular prayers at the two ends ...	...و نماز را در دو طرف روز و ساعات نخستین شب برپا	quran
3	And it came to pass, that, when I was come aga...	...و فرمود تا مدعیانش نزد تو حاضر شوند؛ و از او ب	bible
4	Ah woe, that Day, to the Rejecters of Truth!	!وای در آن روز بر تکذیب کنندگان	quran
...	...	...	...
12595	Women impure are for men impure, and men impur...	...زنان پلید برای مردان پلید و مردان پلید برای زن	quran
12596	I don't want any silly dance given in my honour.'	...بنابراین حالا هم میل ندارم جشنی به افتخار من د	mizan
12597	And the Earth will shine with the Glory of its...	...و زمین به نور پروردگارش روشن می‌شود، و کتاب [ا	quran
12598	Then lifted I up mine eyes, and saw, and behol...	...گفتم: «این چیست؟» او جواب داد: «این است آن ایف	bible
12599	His soul was dried up.	.روح خشکیده بود	mizan

12600 rows × 3 columns

Figure 21-Training Data of question 2&3

```
print(train_df['category'].value_counts())
plt.title("Train dataset value counts")
train_df['category'].value_counts().plot.bar(color='maroon'
```

```
quran    4200
bible    4200
mizan    4200
Name: category, dtype: int64
<matplotlib.axes._subplots.AxesSubplot at 0x7f24edec9710>
```

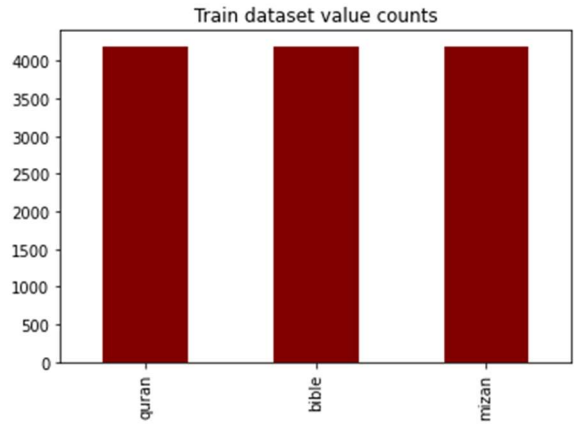


Figure 22-Train data value counts of question 2&3 classes

```
print(test_df['category'].value_counts())
plt.title("Train dataset value counts")
test_df['category'].value_counts().plot.bar(color='maroon')
```

```
mizan    900
bible    900
quran    900
Name: category, dtype: int64
<matplotlib.axes._subplots.AxesSubplot at 0x7f24ee0ef7d0>
```

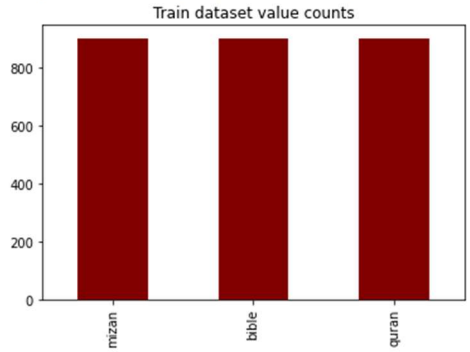


Figure 23-Test data value counts of question 2&3 classes

Pay attention that the datas are balaned.

Part 1

In my network there is a hidden layer with 1024 cells and I added a dropout layer to prevent overfitting. The total procedure is like question1. The ‘bert-beased-case’ checkpoint is used as the embedding layer model in this part. I labeled the ‘category’ column by numbers 0, 1 and 2. All the parameters are as said in the question. The summery of the model:

Layer (type)	Output Shape	Param #	Connected to
input_ids (InputLayer)	[(None, 128)]	0	[]
attention_mask (InputLayer)	[(None, 128)]	0	[]
tf_bert_model (TFBertModel)	TFBaseModelOutputWithPoolingAndCrossAttentions(last_hidden_state=(None, 128, 768), pooler_output=(None, 768), past_key_values=None, hidden_states=None, attentions=None, cross_attentions=None)	108310272	['input_ids[0][0]', 'attention_mask[0][0]']
dropout_37 (Dropout)	(None, 768)	0	['tf_bert_model[0][1]']
dense (Dense)	(None, 1024)	787456	['dropout_37[0][0]']
outputs (Dense)	(None, 3)	3075	['dense[0][0]']

Total params: 109,100,803
Trainable params: 109,100,803
Non-trainable params: 0

Figure 24-Summary of the model in question 2 part 1

I trained the model by 10 epoches:

Part 1

```
[ ] classifier_Bert = Classifier('bert-base-cased', 'source', 'source',  batch_size=32, seq_len=128)
    part1_model = classifier_Bert.create_model()
    part1_model, part1_history, part1_score, part1_predictions = classifier_Bert.train_and_evaluate_model(part1_model)
```

Figure 25-Running the model in python

Here are the results of each epoch:

```
Epoch 1/10
393/393 [==============================] - 354s 875ms/step - loss: 0.1937 - accuracy: 0.9273 - val_loss: 0.0573 - val_accuracy: 0.9818
Epoch 2/10
393/393 [==============================] - 345s 879ms/step - loss: 0.0481 - accuracy: 0.9847 - val_loss: 0.0665 - val_accuracy: 0.9810
Epoch 3/10
393/393 [==============================] - 345s 879ms/step - loss: 0.0248 - accuracy: 0.9929 - val_loss: 0.0610 - val_accuracy: 0.9803
Epoch 4/10
393/393 [==============================] - 345s 879ms/step - loss: 0.0159 - accuracy: 0.9955 - val_loss: 0.1166 - val_accuracy: 0.9725
Epoch 5/10
393/393 [==============================] - 327s 831ms/step - loss: 0.0129 - accuracy: 0.9967 - val_loss: 0.1068 - val_accuracy: 0.9788
Epoch 6/10
393/393 [==============================] - 326s 830ms/step - loss: 0.0081 - accuracy: 0.9975 - val_loss: 0.1161 - val_accuracy: 0.9781
Epoch 7/10
393/393 [==============================] - 345s 878ms/step - loss: 0.0099 - accuracy: 0.9974 - val_loss: 0.0963 - val_accuracy: 0.9818
Epoch 8/10
393/393 [==============================] - 327s 831ms/step - loss: 0.0091 - accuracy: 0.9977 - val_loss: 0.1274 - val_accuracy: 0.9747
Epoch 9/10
393/393 [==============================] - 345s 878ms/step - loss: 0.0102 - accuracy: 0.9972 - val_loss: 0.0838 - val_accuracy: 0.9829
Epoch 10/10
393/393 [==============================] - 326s 830ms/step - loss: 0.0077 - accuracy: 0.9980 - val_loss: 0.0954 - val_accuracy: 0.9814
```

Figure 26-Result of 10 epoches after fitting the model in question 2 part 1

The results are:

```
Loss on test data: 0.10784538835287094
Accuracy on test data: 0.9781481623649597
Classification Report:
              precision    recall  f1-score   support

       quran       1.00      0.95      0.98       900
       bible       0.98      0.99      0.98       900
       mizan       0.96      0.99      0.98       900

    accuracy                           0.98      2700
   macro avg       0.98      0.98      0.98      2700
weighted avg       0.98      0.98      0.98      2700

AUC - quran:  0.02605000000000002
AUC - accuracy:  0.4851314814814815
AUC - change:  0.9888185185185184
```

Figure 27-Result & classification report of the model in question 2 part 1

The plots are as follows:

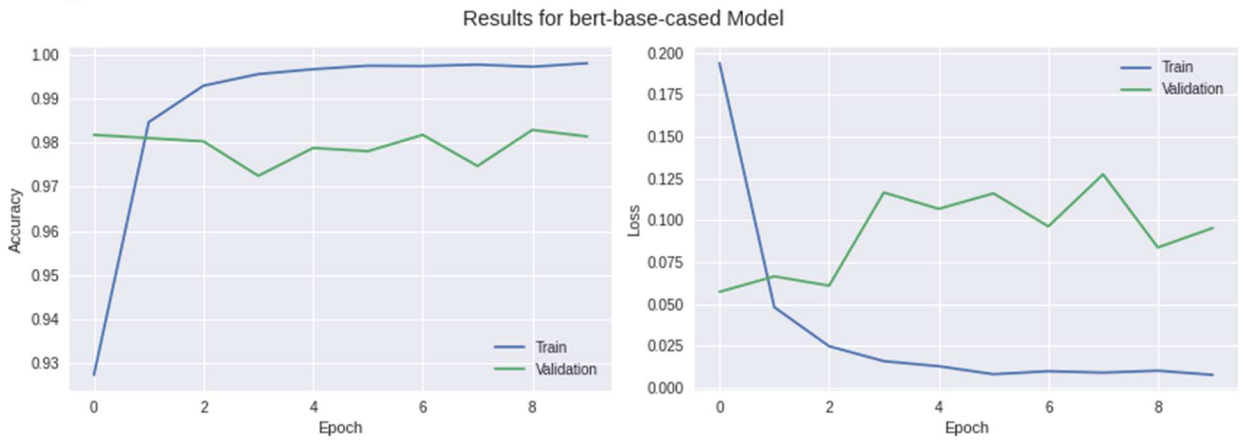


Figure 28-Accuracy & Loss plots in question 2 part 1

The confusion matrix is:

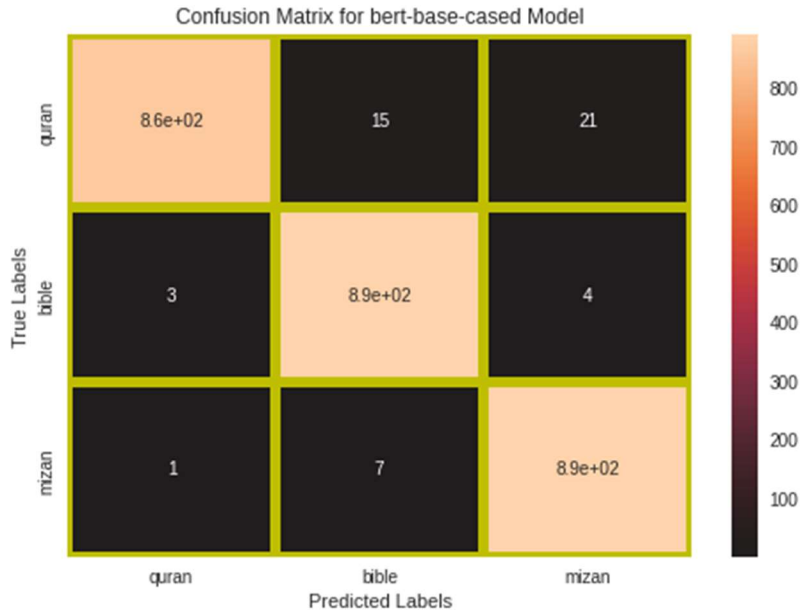


Figure 29-Confusion Matrix in question 2 part 1

As it is shown, the result is **98%** and the model is well designed for this task. Also, I can mention that the test dataset is **balanced** so there is no bias in the classification.

Part 2

In this part we should use the model in part 1, but here the embedding layer is created by ParsBERT model and I used the 'targets' column which is in Persian.

Part 2

```
[ ] classifier_ParsBert = Classifier('HooshvareLab/bert-base-parsbert-uncased', 'targets', 'targets', batch_size=32, seq_len=128)
    part2_model = classifier_ParsBert.create_model()
    part2_model, part2_history, part2_score, part2_predictions = classifier_ParsBert.train_and_evaluate_model(part2_model)
```

Figure 30-Running the model in python

The summery of the model:

Layer (type)	Output Shape	Param #	Connected to
input_ids (InputLayer)	[(None, 128)]	0	[]
attention_mask (InputLayer)	[(None, 128)]	0	[]
tf_bert_model_1 (TFBertModel)	TFBaseModelOutputWithPoolingAndCrossAttentions(last_hidden_state=(None, 128, 768), pooler_output=(None, 768), past_key_values=None, hidden_states=None, attentions=None, cross_attentions=None)	162841344	['input_ids[0][0]', 'attention_mask[0][0]']
dropout_75 (Dropout)	(None, 768)	0	['tf_bert_model_1[0][1]']
dense_1 (Dense)	(None, 1024)	787456	['dropout_75[0][0]']
outputs (Dense)	(None, 3)	3075	['dense_1[0][0]']

Total params: 163,631,875
Trainable params: 163,631,875
Non-trainable params: 0

Figure 31-Summary of the model in question 2 part 2

I trained the model by 10 epoches:

```
Epoch 1/10
393/393 [==============================] - 347s 860ms/step - loss: 0.2221 - accuracy: 0.9175 - val_loss: 0.0942 - val_accuracy: 0.968
Epoch 2/10
393/393 [==============================] - 337s 858ms/step - loss: 0.0582 - accuracy: 0.9803 - val_loss: 0.0904 - val_accuracy: 0.975
Epoch 3/10
393/393 [==============================] - 356s 906ms/step - loss: 0.0269 - accuracy: 0.9909 - val_loss: 0.1450 - val_accuracy: 0.962
Epoch 4/10
393/393 [==============================] - 337s 858ms/step - loss: 0.0149 - accuracy: 0.9957 - val_loss: 0.1216 - val_accuracy: 0.968
Epoch 5/10
393/393 [==============================] - 337s 858ms/step - loss: 0.0102 - accuracy: 0.9967 - val_loss: 0.2299 - val_accuracy: 0.955
Epoch 6/10
393/393 [==============================] - 355s 905ms/step - loss: 0.0086 - accuracy: 0.9976 - val_loss: 0.1408 - val_accuracy: 0.968
Epoch 7/10
393/393 [==============================] - 337s 858ms/step - loss: 0.0117 - accuracy: 0.9962 - val_loss: 0.1678 - val_accuracy: 0.967
Epoch 8/10
393/393 [==============================] - 356s 906ms/step - loss: 0.0069 - accuracy: 0.9976 - val_loss: 0.1863 - val_accuracy: 0.965
Epoch 9/10
393/393 [==============================] - 337s 858ms/step - loss: 0.0079 - accuracy: 0.9980 - val_loss: 0.1463 - val_accuracy: 0.972
Epoch 10/10
393/393 [==============================] - 356s 906ms/step - loss: 0.0064 - accuracy: 0.9986 - val loss: 0.1505 - val accuracy: 0.971
```

Figure 32-Result of 10 epoches after fitting the model in question 2 part 2

The wanted results are:

```
Loss on test data: 0.20684605836868286
Accuracy on test data: 0.965925931930542
Classification Report:
              precision    recall  f1-score   support

       quran       0.96      0.97      0.97       900
       bible       0.98      0.95      0.97       900
       mizan       0.95      0.97      0.96       900

    accuracy                           0.97      2700
   macro avg       0.97      0.97      0.97      2700
weighted avg       0.97      0.97      0.97      2700

AUC - quran:  0.025391975308641953
AUC - accuracy:  0.503608024691358
AUC - change:  0.971
```

Figure 33-Result & classification report of the model in question 2 part 2

Plot:

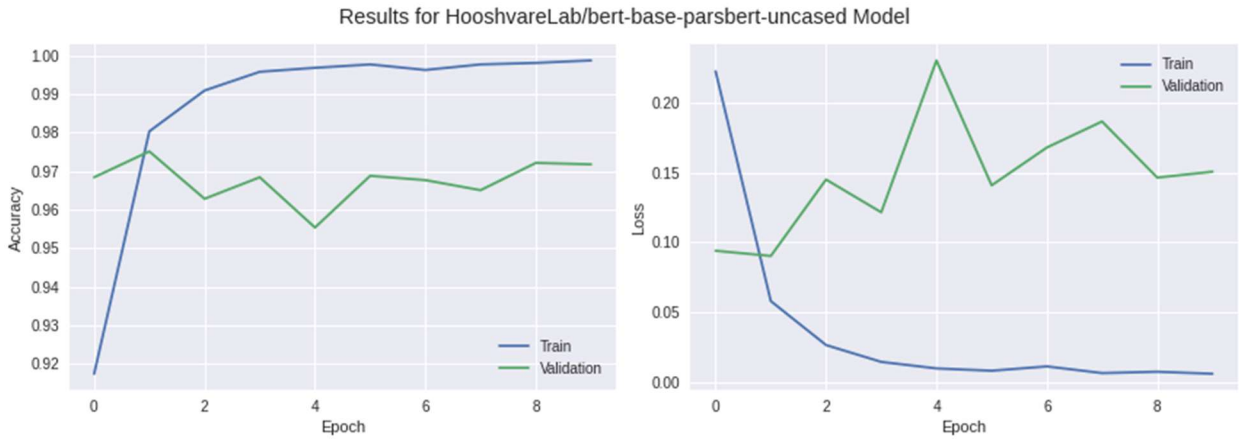


Figure 34-Accuracy & Loss plots in question 2 part 2

The confusion matrix:

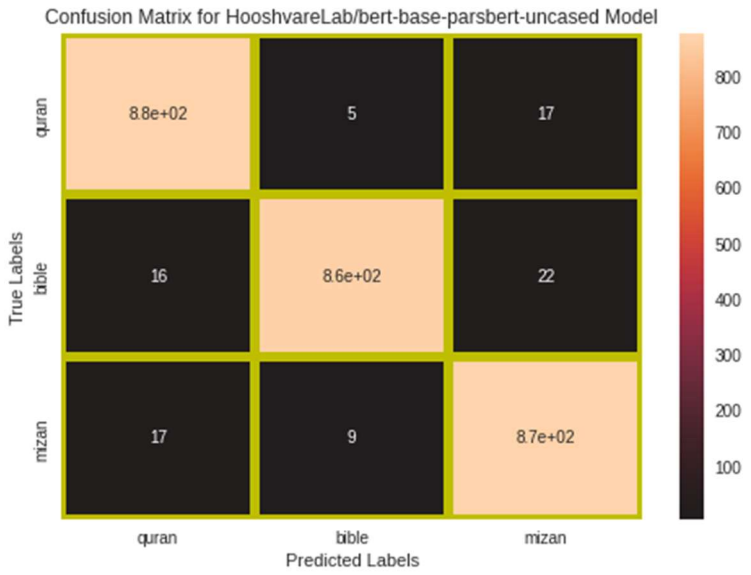


Figure 35-Confusion Matrix in question 2 part 2

The result of this part is 97% which is very good. It is slightly less than the last part(bert-base-cased). We can justify this by considering the vocabulary size and that the Bert model is more **complex** and **generalized** than ParsBert so it could give us better results. I can mention that the **vocabulary size** of ParsBert is less than Bert. Pay attention that Persian is a **morphological-rich** language unlike English.

Part 3

The network used in this part is like part 1 and part 2, the difference is that now we use ‘xlm-roberta-base’ as the embedding model. Also, now we use both ‘source’ and ‘targets’ columns of datasets. To do this we use the encode-plus function of the tokenizer which by passing two sentences ro it, it puts a [SEP] token between the two sentenes and give the input ids and mask ids.

The summery of the model:

```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
input_ids (InputLayer)       [(None, 128)]             0           []

attention_mask (InputLayer)  [(None, 128)]             0           []

tfxlm_roberta_model_2 (TFXLMRo  TFBaseModelOutputWi  278043648  ['input_ids[0][0]',
bertaModel)                  thPoolingAndCrossAt                  'attention_mask[0][0]']
                             tentions(last_hidde
                             n_state=(None, 128,
                              768),
                              pooler_output=(Non
                             e, 768),
                              past_key_values=No
                             ne, hidden_states=N
                             one, attentions=Non
                             e, cross_attentions
                             =None)

dropout_113 (Dropout)        (None, 768)               0           ['tfxlm_roberta_model_2[0][1]']

dense_2 (Dense)              (None, 1024)              787456      ['dropout_113[0][0]']

outputs (Dense)              (None, 3)                 3075        ['dense_2[0][0]']

==================================================================================================
Total params: 278,834,179
Trainable params: 278,834,179
Non-trainable params: 0
```

Figure 36-Summary of the model in question 2 part 3

Part 3

```
multilingualClassifier_XLMR = MultilingualClassifier('xlm-roberta-base', '', '', batch_size=32, seq_len=128)
part3_model = multilingualClassifier_XLMR.create_model()
part3_model, part3_history, part3_score, part3_predictions = multilingualClassifier_XLMR.train_and_evaluate_model(part3_model)
```

Figure 37-Running the model in python

I trained the model by 10 epoches:

```
Epoch 1/10
393/393 [==============================] - 373s 926ms/step - loss: 0.2862 - accuracy: 0.8876 - val_loss: 0.0644 - val_accuracy: 0.9829
Epoch 2/10
393/393 [==============================] - 363s 923ms/step - loss: 0.0510 - accuracy: 0.9862 - val_loss: 0.0352 - val_accuracy: 0.9885
Epoch 3/10
393/393 [==============================] - 362s 922ms/step - loss: 0.0256 - accuracy: 0.9926 - val_loss: 0.0636 - val_accuracy: 0.9851
Epoch 4/10
393/393 [==============================] - 362s 922ms/step - loss: 0.0211 - accuracy: 0.9940 - val_loss: 0.0581 - val_accuracy: 0.9881
Epoch 5/10
393/393 [==============================] - 363s 923ms/step - loss: 0.0183 - accuracy: 0.9950 - val_loss: 0.0342 - val_accuracy: 0.9900
Epoch 6/10
393/393 [==============================] - 382s 972ms/step - loss: 0.0141 - accuracy: 0.9955 - val_loss: 0.0488 - val_accuracy: 0.9881
Epoch 7/10
393/393 [==============================] - 382s 972ms/step - loss: 0.0258 - accuracy: 0.9928 - val_loss: 0.0288 - val_accuracy: 0.9907
Epoch 8/10
393/393 [==============================] - 382s 972ms/step - loss: 0.0142 - accuracy: 0.9957 - val_loss: 0.0273 - val_accuracy: 0.9940
Epoch 9/10
393/393 [==============================] - 382s 973ms/step - loss: 0.0108 - accuracy: 0.9966 - val_loss: 0.0429 - val_accuracy: 0.9892
Epoch 10/10
393/393 [==============================] - 363s 923ms/step - loss: 0.0117 - accuracy: 0.9968 - val_loss: 0.0404 - val_accuracy: 0.9914
```

Figure 38-Result of 10 epoches after fitting the model in question 2 part 3

The results are:

```
Loss on test data: 0.06163453310728073
Accuracy on test data: 0.9911110997200012
Classification Report:
              precision    recall  f1-score   support

       quran       1.00      0.98      0.99       900
       bible       0.99      1.00      1.00       900
       mizan       0.98      1.00      0.99       900

    accuracy                           0.99      2700
   macro avg       0.99      0.99      0.99      2700
weighted avg       0.99      0.99      0.99      2700

AUC - quran:  0.013879629629629606
AUC - bible:  0.491408024691358
AUC - mizan:  0.9947123456790123
```

Figure 39-Result & classification report of the model in question 2 part 3

The plots can be shown here:

Figure 40-Accuracy & Loss plots in question 2 part 3

The confusion matrix is as follows:

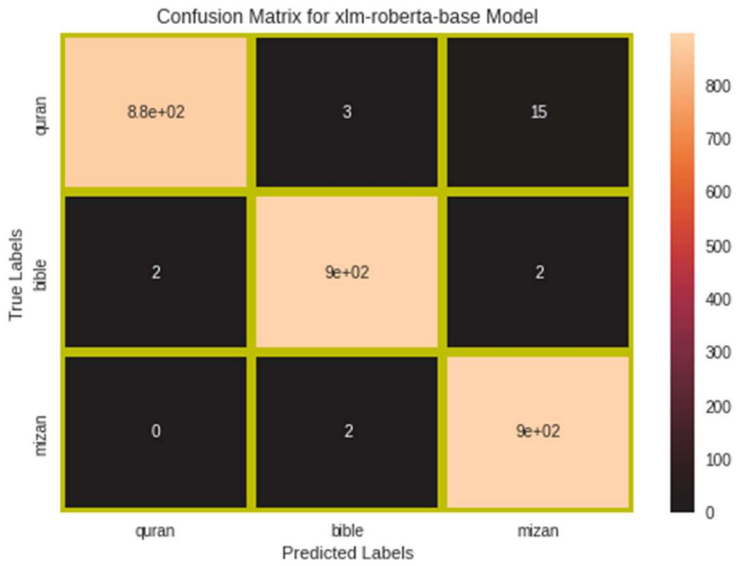


Figure 41-Confusion Matrix in question 2 part 2

- ✓ **My analysis based on the results:** All 3 results are very good but as we saw the multiligual xlm-roberta did better than two other models. And the ParsBert did worst (ParsBert(97%) < Bert(98%) < XLM-RoBERTa(99%)). The bert model which was trained on English dataset has more vocabulary than ParsBert and also Persian is morphological-rich unlike English and also the Bert model is more complex. Hence, all these together we can justify the results.

- ✓ **Did the multilingual model increased the accuracy?** Yes, we got the best result in the third part of this question. Using a multilingual model on multilingual data can increase accuracy. In multilingual model we can consider different aspects of data. For instance we can consider bible in both English and Persian so and this can help the model to classify better. Sometimes features can be extracted better in one language but can not be as good in another one, because there are information in some languages which may not in be in others. For instance, the f1-accuracy of quran by ParsBert is 97% but by xlm-roberta is 99% which means there are some information in English which can help to learn the quran datas better rather than just Persian dataset. Although we got bether results bu multilingual model than monolingual model in this task, we saw the opposite in question 1, So using Bert models is dependent to the task.

Question 3: Cross-lingual zero-shot transfer learning (Bonus)

Cross-lingual transfer refers to transfer learning using data and models available for one language for which ample such resources are available (e.g., English) to solve tasks in another, commonly more low-resource, language. zero-shot learning is a problem setup in machine learning, where at testing, a learner observes samples from classes that were not observed while training the model and predicts the category they belong to.

I used the network and the parameters of the created network in Question 2 with 'xlm-roberta'base' model. In this part I used the 'source' column as the train and validation data (English) and the 'targets' column as the test data (Persian).
I created a class named Classifier in python:

```
class Classifier:
    def __init__(self, model_address, train_column_name, test_column_name, batch_size=32, seq_len=128):
      self.train_df = pd.read_excel('/content/train.xlsx')
      self.valid_df = pd.read_excel('/content/valid.xlsx')
      self.test_df = pd.read_excel('/content/test.xlsx')
      self.encoded_dict = {'quran' : 0, 'bible': 1, 'mizan': 2}
      self.seq_len = seq_len
      self.model_address = model_address
      self.train_column_name = train_column_name
      self.test_column_name = test_column_name
      print(f"train name: {self.train_column_name}")
      print(f"test name: {self.test_column_name }")
      self.batch_size = batch_size
      self.prepare_data()
      self.set_model_and_tokenizer()

    def encode_classes(self, df):
      return df.category.map(self.encoded_dict)

    def prepare_data(self):
      self.train_df['category'] = self.encode_classes(self.train_df)
      self.valid_df['category'] = self.encode_classes(self.valid_df)
      self.test_df['category'] = self.encode_classes(self.test_df)

    def set_model_and_tokenizer(self):
      if self.model_address != 'xlm-roberta-base':
        self.tokenizer = AutoTokenizer.from_pretrained(self.model_address)
        self.bert = TFAutoModel.from_pretrained(self.model_address)
      else:
        self.tokenizer = AutoTokenizer.from_pretrained(self.model_address, from_pt=True)
        self.bert = TFAutoModel.from_pretrained(self.model_address, from_pt=True)
```

Here is the model summary:

```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
input_ids (InputLayer)       [(None, 128)]             0           []

attention_mask (InputLayer)  [(None, 128)]             0           []

tfxlm_roberta_model (TFXLMRobe  TFBaseModelOutputWi  278043648  ['input_ids[0][0]',
rtaModel)                    thPoolingAndCrossAt                 'attention_mask[0][0]']
                             tentions(last_hidde
                             n_state=(None, 128,
                              768),
                              pooler_output=(Non
                             e, 768),
                              past_key_values=No
                             ne, hidden_states=N
                             one, attentions=Non
                             e, cross_attentions
                             =None)

dropout_37 (Dropout)         (None, 768)               0           ['tfxlm_roberta_model[0][1]']

dense (Dense)                (None, 1024)              787456      ['dropout_37[0][0]']

outputs (Dense)              (None, 3)                 3075        ['dense[0][0]']

==================================================================================================
Total params: 278,834,179
Trainable params: 278,834,179
Non-trainable params: 0
```

Figure 42-Summary of the model in question 3

I trained the model by 10 epoches:

```
Epoch 1/10
393/393 [==============================] - 359s 890ms/step - loss: 0.3327 - accuracy: 0.8594 - val_loss: 0.0955 - val_accuracy: 0.9710
Epoch 2/10
393/393 [==============================] - 348s 885ms/step - loss: 0.0879 - accuracy: 0.9726 - val_loss: 0.0797 - val_accuracy: 0.9740
Epoch 3/10
393/393 [==============================] - 348s 885ms/step - loss: 0.0540 - accuracy: 0.9840 - val_loss: 0.0775 - val_accuracy: 0.9766
Epoch 4/10
393/393 [==============================] - 348s 886ms/step - loss: 0.0409 - accuracy: 0.9873 - val_loss: 0.1049 - val_accuracy: 0.9740
Epoch 5/10
393/393 [==============================] - 348s 885ms/step - loss: 0.0218 - accuracy: 0.9939 - val_loss: 0.1046 - val_accuracy: 0.9702
Epoch 6/10
393/393 [==============================] - 348s 885ms/step - loss: 0.0293 - accuracy: 0.9901 - val_loss: 0.1083 - val_accuracy: 0.9766
Epoch 7/10
393/393 [==============================] - 348s 885ms/step - loss: 0.0269 - accuracy: 0.9915 - val_loss: 0.0691 - val_accuracy: 0.9833
Epoch 8/10
393/393 [==============================] - 348s 885ms/step - loss: 0.0126 - accuracy: 0.9965 - val_loss: 0.0712 - val_accuracy: 0.9799
Epoch 9/10
393/393 [==============================] - 348s 885ms/step - loss: 0.0129 - accuracy: 0.9953 - val_loss: 0.0835 - val_accuracy: 0.9792
Epoch 10/10
393/393 [==============================] - 348s 886ms/step - loss: 0.0255 - accuracy: 0.9934 - val_loss: 0.0881 - val_accuracy: 0.9807
```

Figure 43-Result of 10 epoches after fitting the model in question 3

The results are:

```
Loss on test data: 1.5041240453720093
Accuracy on test data: 0.7737036943435669
Classification Report:
              precision    recall  f1-score   support

       quran       0.80      0.74      0.77       900
       bible       0.80      0.60      0.69       900
       mizan       0.74      0.98      0.84       900

    accuracy                           0.77      2700
   macro avg       0.78      0.77      0.77      2700
weighted avg       0.78      0.77      0.77      2700

AUC - quran:  0.1598077160493827
AUC - bible:  0.4312493827160494
AUC - mizan:  0.9089429012345679
```

Figure 44-Result & classification report of the model in question 3

As we can see the results is **77%**.

The plots can be shown here:

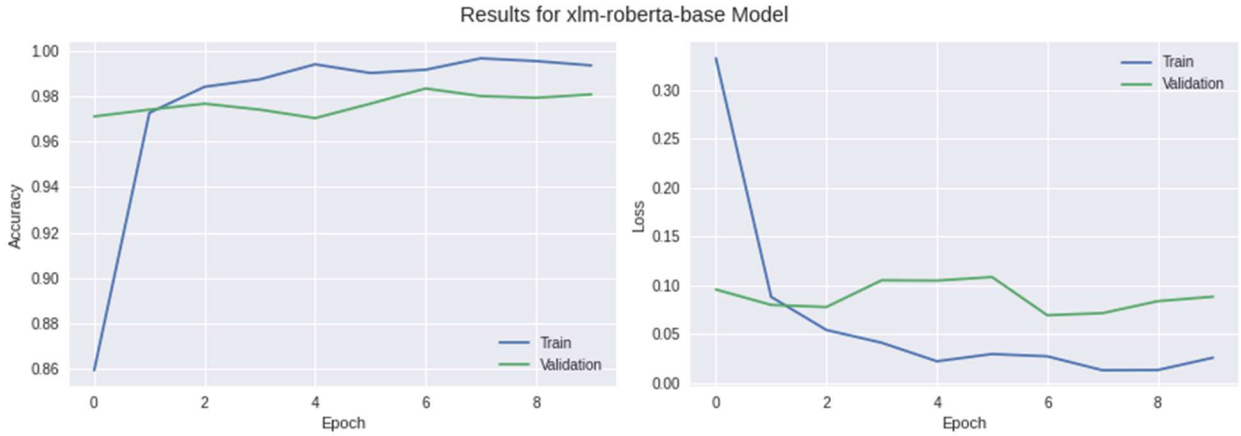


Figure 45-Accuracy & Loss plots in question 3

Also, the confusion matrix is as follows:

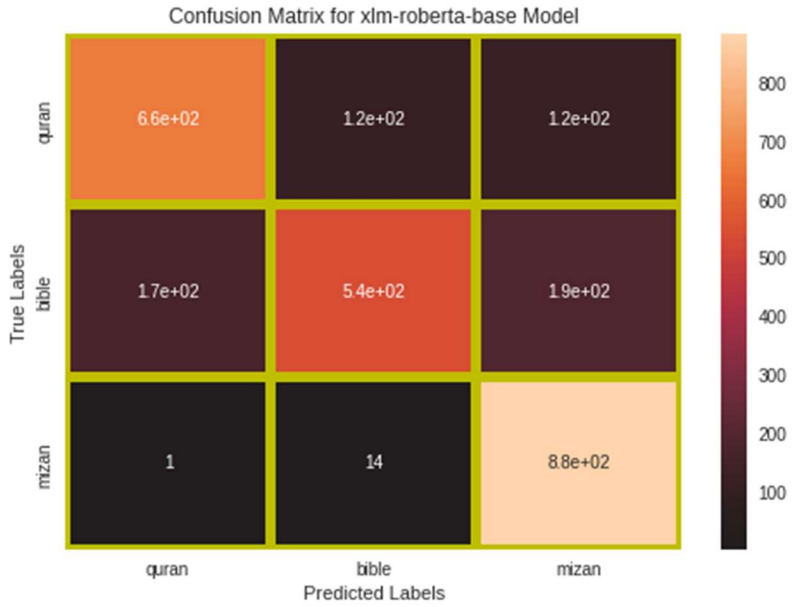


Figure 46-Confusion Matrix in question 3

Part 1

I expect the performance to get lower from results in question 2, since we are training the model on one language and test it on another model. So, the model can't learn as well as models in question 2. Though, I can mention that multilingual Bert models like xlm-roberta can learn words in a lot of languages and also, they can create similar embeddings for words which have same sense in different languages.

Part 2

As I mentioned in part 1, the xlm-roberta model trained on English language, learned different senses ad can create embedding for similar word in Persia. Hence, in our case which the model learned the English data can use this to create acceptable embeddings for Persian. The 77% result is not as good as results in question 2 (98%) which the train and test datasets has the same language but, it is acceptable. Pay attention that since the number of records in the test dataset for each class is the same, if the classification was random we could get 33% of accuracy but we got 77% and this shows that the model could learn the context very well so it has good performance on Persian language (of course not as good as when the train dataset is Perisan).

Also, according to this paper, recent results of different papers show that **multilingual** models exhibit great performance even when evaluated on a single

language. XLM-RoBERTa is trained on a multilingual language modeling **objective** using only monolingual data. This basically means samples of text streams are taken from all the languages, and the model is trained to predict masked tokens in the input. XLM-RoBERTa is a multilingual model trained on 100 different languages. Unlike some XLM multilingual models, it does not require lang tensors to understand which language is used and should be able to determine the correct language from the **input ids**.

Part 3

According to this paper, some NLP tasks are not applicable to most human languages due to the **lack of annotated training data** for various NLP tasks. Cross-lingual transfer learning (CLTL) is a viable method for building NLP models for a **low-resource** target language by leveraging labeled data from other (source) languages which have more labeled data. I can mention that sometimes it is better to have a large labeled data in another language than training the model with small data with the same language as test data.

Resources:
https://arxiv.org/pdf/1810.04805.pdf
https://aclanthology.org/P19-1299.pdf
https://github.com/hooshvare/parsbert
https://arxiv.org/pdf/2103.09519.pdf
https://www.aaai.org/AAAI22Papers/AAAI-2785.JiangX.pdf
https://github.com/yezhengli-Mr9/transformers/blob/master/docs/source/pretrained_models.rst