بنام خدا

دانشکده‌ی مهندسی برق و کامپیوتر

درس پردازش زبان‌های طبیعی

شایان واصف احمدزاده

شماره دانشجویی : 810197603

تمرین کامپیوتری 4

روز آپلود : 4 خرداد **Transformers/BERT** استاد : دکتر فیلی

---

# Table of Contents



*تمامی مدل های آموزش داده شده در سوال 2 و 3 در این لینک قابل[1] دسترسی هستند.

[1] https://drive.google.com/drive/folders/1WfZDY6q2cQLP33NKgxAmrbD2jBWT9DBq?usp=sharing

# ParsiNLU dataset classification

## 1. PreProcessing

در ابتدا کتابخانه‌های لازم را در محیط colab ایجاد می‌کنیم :

```
import transformers
from transformers import BertTokenizer, BertForSequenceClassification
from transformers import XLMRobertaTokenizer, XLMRobertaForSequenceClassif
ication
from transformers import AdamW
```

به دلیل حجم بالای محسابات، از واحد پردازشی GPU به جای CPU استفاده می‌کنیم :

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
```

حال دیتاست داده شده را به کمک کتابخانه ذکر شده[2] load می‌کنیم :

```
from datasets import load_dataset

dataset = load_dataset("persiannlp/parsinlu_entailment")
for x in dataset['train']:
  print(x)
```

در ادامه نمونه‌هایی از جملات این دیتاست را بررسی می‌کنیم :

```
dataset['train'][0]['sent1']
```

زنان به قدری بخش بزرگی از نیروی کار را تشکیل می دهند که به سختی می توان باور داشت که اگر این امر در مورد زنان صادق نباشد ، این امر می تواند صادق باشد.

```
dataset['train'][0]['sent2']
```

مردان بخش عظیمی از نیروی کار هستند بنابراین تنها افراد مهم هستند.

```
len(dataset['train'])
```

755

در ادامه، سه ستون اول را به ترتیب ‘sent1’، ‘sent2’ و ‘label’ نام گذاری می‌کنیم :

```
import pandas as pd
train_df = pd.DataFrame(train_data , columns = ['sent1' , 'sent2' , 'label
'])
test_df = pd.DataFrame(test_data , columns = ['sent1' , 'sent2' , 'label']
)
```

[2] Hugging Face

```
valid_df = pd.DataFrame(valid_data , columns = ['sent1' , 'sent2' , 'label
'])
train_df.head()
```

5 سطر اول دیتاست مربوط به دادگان آموزش به شکل زیر می‌باشد :

| | sent1 | sent2 | label |
|---|---|---|---|
| 0 | ...زنان به قدری بخش بزرگی از نیروی کار را تشکیل م | ...مردان بخش عظیمی از نیروی کار هستند بنابراین تن | c |
| 1 | ...سالها است که کنگره در تلاش است تا اثربخشی مدیر | ...کنگره بودجه ویژه ای برای مدیریت اطلاعات و فناو | n |
| 2 | ...سرامیک‌های زیست خنثی پس از قرارگیری در بدن میز | .خواص فیزیکی سرامیک‌ها قابل اندازه گیری است | n |
| 3 | ...دولت از هیچ قانونی که منجر به کاهش چشمگیر توان | ...قانونی که باعث کاهش استفاده از زغال سنگ به عنو | e |
| 4 | ...روش‌ها و الگوریتم‌های بهینه‌سازی به دو دسته ال | .آمار در دروس مدیریتی نقش مهمی را بازی میکند | n |

Fig.1 Train Data Frame

با بررسی برچسب های داده شده به دادگان، مشاهده می‌کنیم که علاوه بر سه برچسب اصلی موجود، برچسب هایی از نوع 'xx' دیده میشوند که آنها را به شکل زیر از دیتاست حذف می‌کنیم :

```
train_df = train_df[train_df['label']!='xx']
test_df = test_df[test_df['label']!='xx']
valid_df = valid_df[valid_df['label']!='xx']
train_df.shape
```

(754, 3)

همچنین در برچسب های مربوط به دادگان تست، 2 سطر دارای برچسب '-' می‌باشند که آنرا اطلاح می‌کنیم :

```
print(test_df['label'].value_counts())
test_df = test_df[test_df['label']!='-']
print(test_df['label'].value_counts())
```

```
e    610
c    561
n    502
-      2
Name: label, dtype: int64
e    610
c    561
n    502
Name: label, dtype: int64
```

در نهایت برای اینکه بتوانیم دقت مدل را بدست آوریم، نیاز است که نوع برچسب مقدار عددی پیدا کند :

```
      train_df['label'] = train_df['label'].map({'c':0 , 'n' : 1 , 'e' : 2
})
test_df['label'] = test_df['label'].map({'c':0 , 'n' : 1 , 'e' : 2})
valid_df['label'] = valid_df['label'].map({'c':0 , 'n' : 1 , 'e' : 2})
      train_df.head()
```

| | sent1 | sent2 | label |
|---|---|---|---|
| 0 | ...زنان به قدری بخش بزرگی از نیروی کار را تشکیل م | ...مردان بخش عظیمی از نیروی کار هستند بنابراین تن | 0 |
| 1 | ...سالها است که کنگره در تلاش است تا اثربخشی مدیر | ...کنگره بودجه ویژه ای برای مدیریت اطلاعات و فناو | 1 |
| 2 | ...سرامیک‌های زیست خنثی پس از قرارگیری در بدن میز | .خواص فیزیکی سرامیک‌ها قابل اندازه گیری است | 1 |
| 3 | ...دولت از هیچ قانونی که منجر به کاهش چشمگیر توان | ...قانونی که باعث کاهش استفاده از زغال سنگ به عنو | 2 |
| 4 | ...روش‌ها و الگوریتم‌های بهینه‌سازی به دو دسته ال | .آمار در دروس مدیریتی نقش مهمی را بازی میکند | 1 |

Fig.2 Train Data Frame(Numeric Labels)

همچنین تحلیل ما باید بر روی جمله هایی با طول بزگتر از صفر انجام گیرد که به همین منظور این مهم را بررسی می‌کنیم :

```
train_df = train_df[(train_df['sent1'].str.split().str.len() > 0) & (train
_df['sent2'].str.split().str.len() > 0)]
valid_df = valid_df[(valid_df['sent1'].str.split().str.len() > 0) & (valid
_df['sent2'].str.split().str.len() > 0)]
```

## XlmRoberta .2

در ابتدا نوع مدل را برای تعریف Tokenizer مشخص می‌کنیم :

```
MODEL_TYPE = 'xlm-roberta-base'

tokenizer = XLMRobertaTokenizer.from_pretrained(MODEL_TYPE)
len(tokenizer)
```

250002

حال نوع Tokenize یک جمله دلخواه را توسط Tokenizer مربوط به مدل xlm-roberta مشاهده می‌کنیم :

```
tokens = tokenizer.tokenize('Heyy There!! See some boys are playing in rai
n')
```

```
print(tokens)
```

['▁Hey', 'y', '▁There', '!!', '▁See', '▁some', '▁boys', '▁are', '▁playing', '▁in', '▁rain']

همچنین می‌توان اندیس متناظر با هر Token را پیدا کرد :

```
indexes = tokenizer.convert_tokens_to_ids(tokens)

print(indexes)
```

[28240, 53, 8622, 1146, 6872, 3060, 115982, 621, 75169, 23, 102044]

در Tokenizer مربوط به هر مدل، تعدادی Token مخصوص برای آماده سازی ورودی به شبکه BERT تعریف شده است :

```
tokenizer.special_tokens_map
```

{'bos_token': '<s>',
 'cls_token': '<s>',
 'eos_token': '</s>',
 'mask_token': '<mask>',
 'pad_token': '<pad>',
 'sep_token': '</s>',
 'unk_token': '<unk>'}

در زیر اندیس ای مربوط به Token های مخصوص را مشاهده می‌کنید :

```
print('bos_token_id <s>:', tokenizer.bos_token_id)
print('eos_token_id </s>:', tokenizer.eos_token_id)
print('sep_token_id </s>:', tokenizer.sep_token_id)
print('pad_token_id <pad>:', tokenizer.pad_token_id)
```

bos_token_id <s>: 0
eos_token_id </s>: 2
sep_token_id </s>: 2
pad_token_id <pad>: 1

به کمک تابع Encode_plus می‌توان نمایش جدید ورودی به شبکه ، اندیس ها و نحوه tokenization و نوع Mask شدن جمله یا دو جمله ورودی را مشاهد کرد. Attention_mask خروجی در واقع نشان می‌دهد که کدام token ها Pad شده هستند و کدام جزو اصلی جمله هستند :

```
MAX_LEN = 15

sentence1 = 'Hello there.'
sentence2 = 'How are you?'
```

```
encoded_dict = tokenizer.encode_plus(
            sentence1, sentence2,
            add_special_tokens = True,
            max_length = MAX_LEN,
            pad_to_max_length = True,
            return_attention_mask = True,
            truncation=True,
            return_tensors = 'pt' # return pytorch tensors
       )


encoded_dict
```

{'input_ids': tensor([[ 0, 35378, 2685, 5, 2, 2, 11249, 621, 398, 32,
2, 1, 1, 1, 1]]), 'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0]])}

طبیعتا خروجی بدست آمده از Attention_mask به حداکثر طول تعیین شده برای هر جمله مربوط می‌باشد که با پارامتر Max_length در مدل تنظیم می‌شود.

```
tokenizer.tokenize(sentence1)
```

['▁Hello', '▁there', '.']

```
tokenizer.tokenize(sentence2)
```

['▁How', '▁are', '▁you', '?']

در زیر نحوه نمایش جدید ورودی را در دو حالت وجود Token های خاص و نبود آن بررسی می‌کنیم :

```
a = tokenizer.decode(input_ids,
                skip_special_tokens=False)

b = tokenizer.decode(input_ids,
                skip_special_tokens=True)

print(a)
print(b)
```

<s[3]> Hello there.</s[4]></s[5]> How are you?</s><pad[6]><pad><pad><pad>
Hello there. How are you?

[3] BOS (Begin of The Sentence)
[4] EOS (End of The Sentence)
[5] SEP(Separator)
[6] PAD

همینطور شرایط مشابه را برای یک جمله هم می‌توان داشت. در واقع در نهایت ورودی ما به شبکه BERT یک جمله پیوسته با Token های مخصوص اضافه شده است.

*بنابراین کاربرد دو جمله در Task های Entailment یا کاربرد های چندزبانه[7] و کاربرد یک جمله در Task های تک زبانه[8] می‌باشد.

```
MAX_LEN = 15

sentence1 = 'Hello there. How are you? Have a nice day. This is a test?'


encoded_dict = tokenizer.encode_plus(
            sentence1,
            max_length = MAX_LEN,
            stride=0,
            pad_to_max_length = True,
            return_overflowing_tokens=True,
            truncation=True
       )


encoded_dict
```

{'overflowing_tokens': [83, 10, 3034, 32], 'num_truncated_tokens': 4, 'input_ids': [0, 35378, 2685, 5, 11249, 621, 398, 32, 31901, 10, 26267, 5155, 5, 3293, 2], 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}

در زیر تنظیمات مربوط به آموزش مدل را آورده‌ایم :

```
MODEL_TYPE = 'xlm-roberta-base'

L_RATE = 5e-4
MAX_LEN = 15

NUM_EPOCHS = 3
BATCH_SIZE = 16
NUM_CORES = os.cpu_count()
```

در ادامه کلاس های مربوط به دیتاست های آموزش، ارزیابی و تست را ایجاد می‌کنیم و به کمک Data Loader ها آنها را به Batch های به طول مساوی تقسیم می‌کنیم که تعداد Batch ها در هر کدام از موارد زیر آورده شده است :

[7] Multilingual
[8] Monolingual

```
print(len(train_dataloader))
print(len(val_dataloader))
print(len(test_dataloader))
```

48
17
105

همچنین در ادامه محتوای هر کدام از Batch ها را بررسی می‌کنیم :

```
# Get one train batch
import transformers
transformers.logging.set_verbosity_error()
padded_token_list, att_mask, target = next(iter(train_dataloader))

print(padded_token_list.shape)
print(att_mask.shape)
print(target.shape)
```

torch.Size([16, 15])
torch.Size([16, 15])
torch.Size([16])

حال به کمک Sequence Classification موجود در Hugging Face، مدل را ایجاد می‌کنیم. نحوه ساخت مدل توسط sequence Classification و ایجاد کلاس یکسان می‌باشد و در واقع بعد از 12 لایه موجود در BERT یک لایه Classification اضافه کرده است که در زیر تنها مشخصات لایه Classification آورده شده است :

```
from transformers import XLMRobertaForSequenceClassification,AdamW

model = XLMRobertaForSequenceClassification.from_pretrained(
    MODEL_TYPE,
    num_labels = 3, # The number of output labels. 2 for binary classifica
tion.
)

# Send the model to the device.
model.to(device)
```

(classifier): RobertaClassificationHead(
  (dense): Linear(in_features=768, out_features=768, bias=True)
  (dropout): Dropout(p=0.1, inplace=False)
  (out_proj): Linear(in_features=768, out_features=3, bias=True)
 )
)

در نهایت بهینه ساز[9] مدل را تعریف می‌کنیم :

```
# Define the optimizer
optimizer = AdamW(model.parameters(),
              lr = L_RATE,
              eps = 1e-8
            )
```

خطای مدل بر روی دادگان آموزش و ارزیابی و دقت مدل بر روی دادگان ارزیابی در طی سه epoch به شکل زیر می‌باشد :

```
======== Epoch 1 / 3 ========
Training...
Train loss: 56.40007174015045

Validation...
Val loss: 19.188144207000732
Val acc:  0.28888888888888886

======== Epoch 2 / 3 ========
Training...
Train loss: 53.47912639379501

Validation...
Val loss: 18.593933582305908
Val acc:  0.40370370370370373

======== Epoch 3 / 3 ========
Training...
Train loss: 53.318181693553925

Validation...
Val loss: 18.956595420837402
Val acc:  0.40370370370370373
```

در نهایت پیشبینی مدل را روی دادگان تست در زیر می‌بینیم :

```
stacked_preds
```

```
array([[ 0.4616568 , -0.33336   ,  0.11875859],
       [ 0.46165755, -0.33336022,  0.11875866],
       [ 0.46165702, -0.33336   ,  0.11875853],
       ...,
       [ 0.4616572 , -0.33336008,  0.11875856],
       [ 0.4616584 , -0.3333608 ,  0.11875865],
       [ 0.461657  , -0.33336005,  0.11875859]], dtype=float32)
```

[9] Optimizer

همانطور که مشخص است، مدل برای تمامی دادگان تست، یک نوع احتمال را پیشبینی کرده و همچنین در زیر می‌بینیم که تمامی برچسب های پیشبین شده عدد صفر هستند :

```
preds = np.argmax(stacked_preds, axis=1)

preds
set(preds)
```

{0}

```
test_df['label'].value_counts()
```

2 610
0 561
1 502
Name: label, dtype: int64

که طبق توزیع برچسب ها، دقت بر روی دادگان تست برابر حدود 33 درصد می‌باشد.

## 3. ParsBERT

در این قسمت نحوه پیاده سازی ParsBERT با مدل بخش قبل کمی تفاوت دارد، بنابراین تفاوت های ایجاد شده را می‌آوریم.

در ابتدا یک تناظر[10] بین برچسب های و مقادیر عددی در قالب Dictionary ایجاد می‌کنیم :

```
      # create a key finder based on label 2 id and id to label

label2id = {label: i for i, label in enumerate(labels)}
id2label = {v: k for k, v in label2id.items()}

print(f'label2id: {label2id}')
print(f'id2label: {id2label}')
```

label2id: {'c': 0, 'e': 1, 'n': 2}
id2label: {0: 'c', 1: 'e', 2: 'n'}

مانند قسمت قبل، Tokenizer مربوط به مدل را ایجاد می‌کنیم و سپس پیکربندی مدل[11] را مانند زیر ایجاد می‌کنیم :

```
# setup the tokenizer and configuration
MODEL_TYPE = "HooshvareLab/bert-base-parsbert-uncased"
tokenizer = BertTokenizer.from_pretrained(MODEL_TYPE)
config = BertConfig.from_pretrained(
    MODEL_TYPE, **{
```

[10] Mapping
[11] Configuration

```
        'label2id': label2id,
        'id2label': id2label,
    })

print(config.to_json_string())
```


```
{
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "id2label": {
    "0": "c",
    "1": "e",
    "2": "n"
  },
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "label2id": {
    "c": 0,
    "e": 1,
    "n": 2
  },
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "transformers_version": "4.19.2",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 100000
}
```


حال اینبار کلاس Textual Entailment را ایجاد می‌کنیم :

```
class Textual_Entailement(nn.Module):

    def __init__(self, config):
        super(Textual_Entailement, self).__init__()
```

```
        self.bert = BertModel.from_pretrained(MODEL_TYPE,return_dict=Fals)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.classifier = nn.Linear(config.hidden_size, config.num_labels)

    def forward(self, input_ids, attention_mask, token_type_ids):
        _, pooled_output = self.bert(
            input_ids=input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids)

        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)
        return logits
```

همانطور که مشخص است، بعد از لایه‌ی BERT، یک لایه Dropout و یه لایه Feedforward اضافه می‌کنیم.

به منظور اینکه از حجم RAM و GPU بهینه استفاده کنیم، حافظه‌ی Cache مربوط به GPU را خالی می‌کنیم :

```
import torch, gc

gc.collect()
torch.cuda.empty_cache()
pt_model = None

!nvidia-smi
```

روند آموزش مدل و دقت و خطا بر روی دادگان آموزش و ارزیابی به صورت می‌باشد :

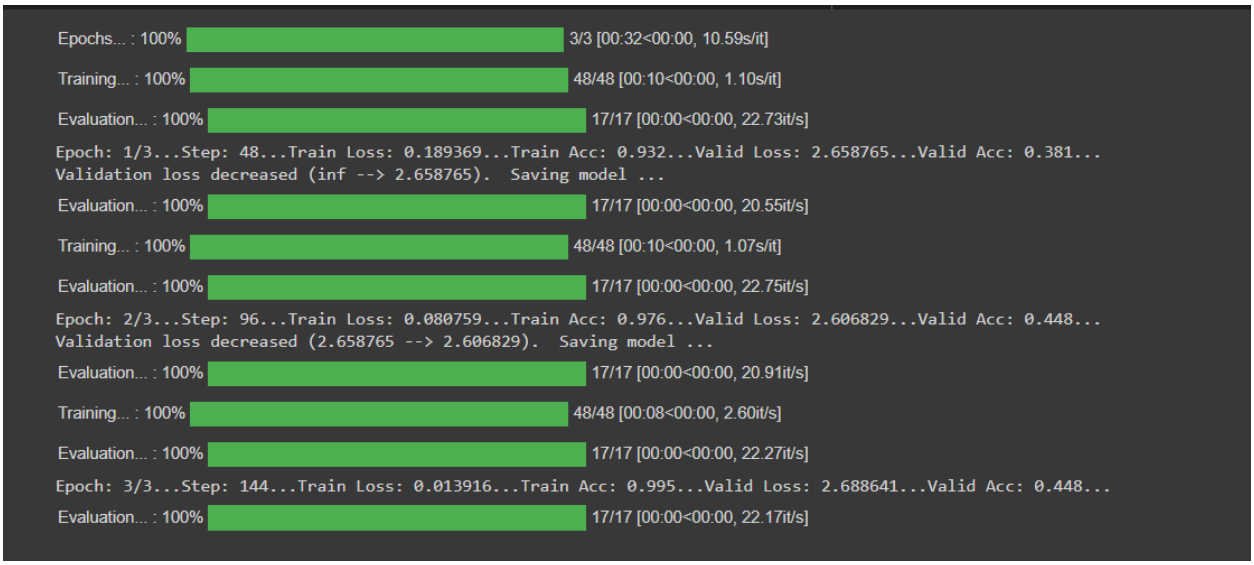


Fig.3 Accuracy / Loss ParsBERT

```
y_test, y_pred = [labels.index(label) for label in test_df['label'].values
], preds

print(f'F1: {f1_score(y_test, y_pred, average="weighted")}')
print()
print(classification_report(y_test, y_pred, target_names=labels))
```

در نهایت گزارش طبقه بند بر روی دادگان تست به صورت زیر می‌باشد :

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| c | 0.42 | 0.26 | 0.32 | 561 |
| e | 0.43 | 0.56 | 0.49 | 610 |
| n | 0.41 | 0.44 | 0.43 | 502 |
| accuracy | | | 0.42 | 1673 |
| macro avg | 0.42 | 0.42 | 0.41 | 1673 |
| weighted avg | 0.42 | 0.42 | 0.41 | 1673 |

همانطور که مشاهده می‌شود، عملکرد مدل بر روی دادگان تست وقتی که از مدل ParsBERT استفاده می‌کنیم بهتر است زیرا مدل ParsBERT بر روی دادگان فارسی آموزش داده شده است و دیتاست ما نیز تماما فارسی می‌باشد پس طبیعتا مدل ParsBERT باید بتواند Embedding های مناسب تری نسبت به Roberta به ما بدهد و در نتیجه بر روی دادگان تست عمکرد بهتری دارد(عملکرد Roberta بر روی دادگان تست رندوم بود ).

*همچنین یکی از محدودیت های این سوال این بود که امکان بررسی همه Hyper parameter ها نبود زیرا از یک جایی به بعد مدل آنقدر بزرگ میشد که در محیط Colab به اصطلاح Crash می‌شد. برای همین دقت بدست آمده برای هر دو مدل محدود به پیچدگی مدل نیز شده است و دقت بدست آمده بهترین عملکرد مدل نیست.

# Multilingual Classification

## 1. Mono-Lingual-BERT

در ابتدا دیتاست های داده شده توسط سوال را می‌خوانیم :

```
import pandas as pd
train_df = pd.read_excel('train.xlsx')
valid_df = pd.read_excel('valid.xlsx')
test_df = pd.read_excel('test.xlsx')
train_df.head()
```

| | 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 |

Fig.4 Train DataFrame Outlook

در ادامه تعداد نمونه از دیتاست خوانده شده را بررسی می‌کنیم :

```
train_df['source'][0]
```

When news is brought to one of them, of (the birth of) a female (child), his face darkens, and he is filled with inward grief!

```
train_df['targets'][0]
```

و چون یکی از آنان را به [ولادت] دختر مژده دهند [از شدت خشم] چهره‌اش سیاه گردد، ودرونش از غصه واندوه لبریز و آکنده شود!!

```
len(train_df)
```

12600

همچنین تعداد و نوع برچسب ها را در هر یک از دیتاست های آموزش، ارزیابی و تست بررسی می‌کنیم تا در صورت نیاز پیش پردازشی انجام دهیم :

```
train_df['category'].value_counts()
```

quran 4200
bible 4200
mizan 4200
Name: category, dtype: int64

```
test_df['category'].value_counts()
```

mizan 900
bible 900
quran 900
Name: category, dtype: int64

```
valid_df['category'].value_counts()
```

bible 900
quran 900
mizan 900
Name: category, dtype: int64

همچنین مانند سوال اول، مقدار برچسب ها باید به مقادیر عددی تبدیل شوند :

```
train_df['category'] = train_df['category'].map({'quran':0 , 'bible' : 1 ,
 'mizan' : 2})
test_df['category'] = test_df['category'].map({'quran':0 , 'bible' : 1 , '
mizan' : 2})
valid_df['category'] = valid_df['category'].map({'quran':0 , 'bible' : 1 ,
 'mizan' : 2})
train_df.head()
```

| | 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 | 0 |
| 1 | After them repaired Zadok the son of Immer ove... | ...و چون دشمنان ما شنیدند که ما آگاه شده‌ایم و خد | 1 |
| 2 | And establish regular prayers at the two ends ... | ...و نماز را در دو طرف روز و ساعات نخستین شب برپا | 0 |
| 3 | And it came to pass, that, when I was come aga... | ...و فرمود تا مدعیانش نزد تو حاضر شوند؛ و از او ب | 1 |
| 4 | Ah woe, that Day, to the Rejecters of Truth! | ای‌وای در آن روز بر تکذیب کنندگان | 0 |

Fig.5 Train DataFrame outlook

برای این قسمت، از مدل BERT استفاده می‌کنیم. ابتدا Tokenizer مربوط به این مدل را ایجاد می‌کنیم :

```
from transformers import BertTokenizer

# Load the BERT tokenizer.
print('Loading BERT tokenizer...')
MODEL_TYPE = 'bert-base-uncased'
tokenizer = BertTokenizer.from_pretrained(MODEL_TYPE, do_lower_case=True)
len(tokenizer)
```

30522

برای مثال، اثر اعمال Tokenizer ایجاد شده را بر روی یک جمله دلخواه می‌بینیم :

```
tokens = tokenizer.tokenize('Heyy There!! See some boys are playing in rai
n')
```

```
print(tokens)
```

['hey', '##y', 'there', '!', '!', 'see', 'some', 'boys', 'are', 'playing', 'in', 'rain']

همینطور اندیس های متناظر با Token های بالا به صورت زیر می‌باشد :

```
indexes = tokenizer.convert_tokens_to_ids(tokens)

print(indexes)
```

[4931, 2100, 2045, 999, 999, 2156, 2070, 3337, 2024, 2652, 1999, 4542]

همچنین Token های مخصوص استفاده شده در مدل BERT به صورت زیر می‌باشد :

```
tokenizer.special_tokens_map
```

{'cls_token': '[CLS]',
 'mask_token': '[MASK]',
 'pad_token': '[PAD]',
 'sep_token': '[SEP]',
 'unk_token': '[UNK]'}

نتیجه خروجی تابع Encode_plus را بر روی همان دو جمله سوال 1 ولی اینبار بر روی Tokenizer مربوط به مدل BERT بررسی می‌کنیم :

```
MAX_LEN = 15

sentence1 = 'Hello there.'
sentence2 = 'How are you?'

encoded_dict = tokenizer.encode_plus(
            sentence1, sentence2,
            add_special_tokens = True,
            max_length = MAX_LEN,
            pad_to_max_length = True,
            return_attention_mask = True,
            truncation=True,
            return_tensors = 'pt' # return pytorch tensors
       )
encoded_dict
```

{'input_ids': tensor([[ 101, 7592, 2045, 1012,  102, 2129, 2024, 2017, 1029,  102,    0,    0,    0,    0,
0]]),
'token_type_ids': tensor([[0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0]]),
'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0]])}

```
a = tokenizer.decode(input_ids,
                skip_special_tokens=False)

b = tokenizer.decode(input_ids,
                skip_special_tokens=True)

print(a)
print(b)
```

[CLS] hello there. [SEP] how are you? [SEP] [PAD] [PAD] [PAD] [PAD] [PAD]
hello there. how are you?

همانطور که مشخص است، نحوه ایجاد جمله جدید جهت ورودی به شبکه BERT متفاوت از مدل Roberta می‌باشد :

پارامتر های مورد نیاز جهت آموزش مدل به صورت زیر تعریف شده‌اند :

```
MODEL_TYPE = 'bert-base-uncased'


L_RATE = 3e-5
MAX_LEN = 128

NUM_EPOCHS = 3
BATCH_SIZE = 32
NUM_CORES = os.cpu_count()

NUM_CORES
#torch.cuda.memory_summary(device=None, abbreviated=False)
```

همینطور تعداد Batch های مربوط به هر یک از Data Loader های آموزش، ارزیابی و تست در زیر آورده شده است :

print(len(train_dataloader)): 394

print(len(val_dataloader)): 85

print(len(test_dataloader)): 85

همینطور محتوای هر کدام از Batch ها در زیر قابل مشاهده است :

```
# Get one train batch
import transformers
transformers.logging.set_verbosity_error()
padded_token_list, att_mask, target = next(iter(train_dataloader))

print(padded_token_list.shape)
print(att_mask.shape)
print(target.shape)
```

torch.Size([32, 128])
torch.Size([32, 128])
torch.Size([32])

به کمک sequence classification بحث شده در سوال اول، اینبار مدل را برای شبکه BERT ایجاد می‌کنیم :

```
model = BertForSequenceClassification.from_pretrained(
    MODEL_TYPE,
    num_labels = 3,
    output_attentions = False,
    output_hidden_states = False)

# Send the model to the device.
model.to(device)
```

و در آخر بهینه ساز مورد نظر را تعریف می‌کنیم :

```
# Define the optimizer
optimizer = AdamW(model.parameters(),
              lr = L_RATE,
              eps = 1e-8
            )
```

======== Epoch 1 / 3 ========
Training...
Train loss: 83.6929615046829

Validation...
Val loss: 8.274173361016437
Val acc:  0.9733333333333334

======== Epoch 2 / 3 ========
Training...

Train loss: 22.48220003710594

Validation...
Val loss: 8.39695893402677
Val acc:  0.9762962962962963

======== Epoch 3 / 3 ========
Training...
Train loss: 8.157585276087048

Validation...
Val loss: 11.871339761215495
Val acc:  0.9725925925925926

حال بعد از آموزش مدل، خروجی های پیشبینی شده را بدست می‌آوریم :

```
# Take the argmax. This returns the column index of the max value in each
row.

preds = np.argmax(stacked_preds, axis=1)

preds
```

array([2, 1, 1, ..., 0, 2, 1])

در نهایت گزار طبقه بند مدل را بر روی دادگان تست بدست می‌آوریم :

F1: 0.9770305731501718

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| quran | 0.98 | 0.97 | 0.97 | 900 |
| bible | 0.99 | 0.96 | 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 |

برای اینکه بتوانیم AUC را بدست آوریم، نیاز است که برچسب های واقی مدل[12] را به صورت One-Hot در آوریم :

```
nb_classes = 3
one_hot_targets = np.eye(nb_classes)[y_test.reshape(-1,1)]
one_hot_targets.reshape((len(y_test),-1))
```

```
array([[0., 0., 1.],
       [0., 1., 0.],
       [0., 1., 0.],
       ...,
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.]])
```

حال مقدار AUC مدل را بر روی دادگان تست محاسبه می‌کنیم :

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score,roc_curve
roc_auc_score(one_hot_targets.reshape((len(y_test),-
1)), torch.softmax(torch.tensor(stacked_preds),dim=1).numpy(), multi_class
='ovo', average='weighted')#, labels=['quran' , 'bible'  , 'mizan'])
```

```
0.9976180041152264
```

همچنین نمودار ROC[13] را برای هر کلاس در برابر کلاس های دیگر[14] رسم می‌کنیم :

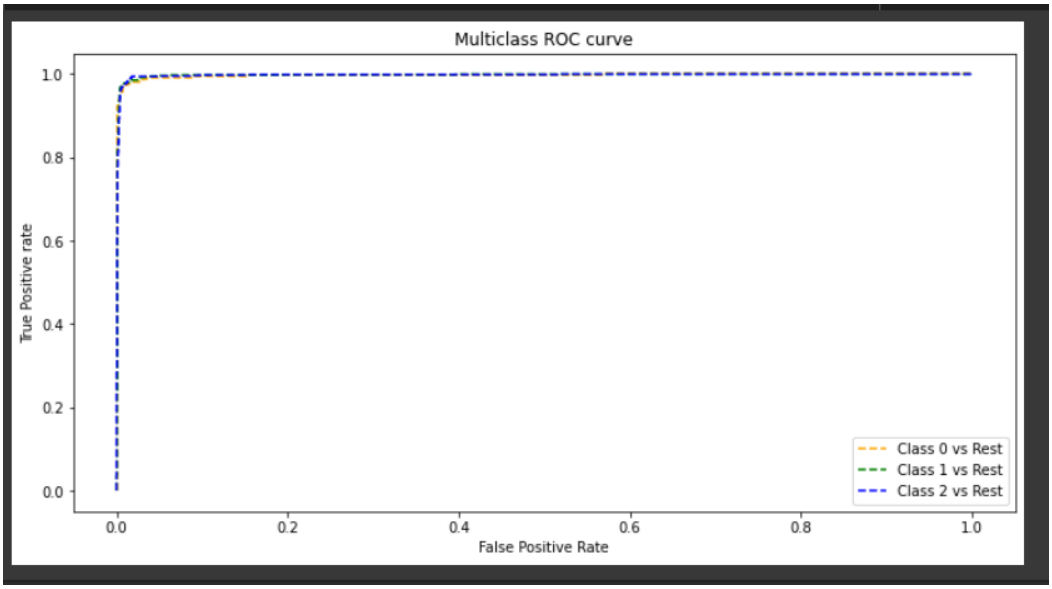


Fig.6 ROC Curve For Multi Classes(English set)

[12] True Labels
[13] ROC Curve
[14] One vs All

## 2. Mono-Lingual-ParsBERT

مانند پیاده سازی ParsBERT در سوال اول قسمت دوم عمل می‌کنیم با این تفاوت که اسم برچسب ها متفاوت می‌باشد :

```
# create a key finder based on label 2 id and id to label

label2id = {label: i for i, label in enumerate(labels)}
id2label = {v: k for k, v in label2id.items()}

print(f'label2id: {label2id}')
print(f'id2label: {id2label}')
```

label2id: {'bible': 0, 'mizan': 1, 'quran': 2}
id2label: {0: 'bible', 1: 'mizan', 2: 'quran'}

مانند قسمت قبل، Tokenizer مربوط به مدل را ایجاد می‌کنیم و سپس پیکربندی مدل را مانند زیر ایجاد می‌کنیم :

```
MODEL_TYPE = "HooshvareLab/bert-base-parsbert-uncased"
tokenizer = BertTokenizer.from_pretrained(MODEL_TYPE)
config = BertConfig.from_pretrained(
    MODEL_TYPE, **{
        'label2id': label2id,
        'id2label': id2label,
    })
```


{
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "id2label": {
    "0": "bible",
    "1": "mizan",
    "2": "quran"
  },
  "initializer_range": 0.02,
  "intermediate_size": 3072,

```
  "label2id": {
    "bible": 0,
    "mizan": 1,
    "quran": 2
  },
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "transformers_version": "4.19.2",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 100000
}
```


همینطور نحوه ایجاد مدل مانند سابق می‌باشد :

```
pt_model = Textual_Entailement(config=config)
pt_model = pt_model.to(device)

print('pt_model', type(pt_model))
```

```
Downloading: 100%
624M/624M [00:24<00:00, 34.8MB/s]
pt_model <class '__main__.Textual_Entailement'>
```

در نهایت، فرآیند آموزش مدل و دقت و خطا برای دادگان آموزش، ارزیابی و تست آورده شده است :

```
Epochs... : 100% 3/3 [06:49<00:00, 136.46s/it]
Training... : 100% 788/788 [02:10<00:00, 2.77s/it]
Evaluation... : 100% 169/169 [00:06<00:00, 26.03it/s]
Epoch: 1/3...Step: 788...Train Loss: 0.155857...Train Acc: 0.943...Valid Loss: 0.072285...Valid Acc: 0.973...
Validation loss decreased (inf --> 0.072285).  Saving model ...
Evaluation... : 100% 169/169 [00:06<00:00, 26.43it/s]
Training... : 100% 788/788 [02:09<00:00, 2.11s/it]
Evaluation... : 100% 169/169 [00:06<00:00, 26.07it/s]
Epoch: 2/3...Step: 1576...Train Loss: 0.021465...Train Acc: 0.993...Valid Loss: 0.077117...Valid Acc: 0.975...
Evaluation... : 100% 169/169 [00:06<00:00, 26.21it/s]
Training... : 100% 788/788 [02:09<00:00, 2.18s/it]
Evaluation... : 100% 169/169 [00:06<00:00, 26.06it/s]
Epoch: 3/3...Step: 2364...Train Loss: 0.004038...Train Acc: 0.999...Valid Loss: 0.082153...Valid Acc: 0.975...
Evaluation... : 100% 169/169 [00:06<00:00, 26.06it/s]
```

در زیر دو آرایه برچسب های پیشبینی شده و احتمال متناظر با هر بر چسب را بدست آورده‌ایم :

```
test_sent1 = test_df['targets'].to_numpy()
preds, probs = predict(pt_model, test_sent1, tokenizer, max_len=32)

print(preds.shape, probs.shape)
```

(2700,) (2700, 3)

```
probs
```

```
array([[1.6039739e-04, 9.9970990e-01, 1.2970810e-04],
       [9.9970728e-01, 1.1518021e-04, 1.7761067e-04],
       [9.9963093e-01, 2.7519450e-04, 9.3868570e-05],
       ...,
       [9.8997436e-05, 4.9335253e-05, 9.9985158e-01],
       [1.5673912e-04, 9.9945492e-01, 3.8831757e-04],
       [9.9961144e-01, 2.5615757e-04, 1.3234555e-04]], dtype=float32)
```

همچنین گزارش طبقه بند مدل بر روی دادگان تست به صورت زیر می‌باشد :

F1: 0.9737040075706846

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| bible | 0.97 | 0.98 | 0.98 | 900 |
| mizan | 0.97 | 0.97 | 0.97 | 900 |
| quran | 0.98 | 0.97 | 0.97 | 900 |
| accuracy | | | 0.97 | 2700 |
| macro avg | 0.97 | 0.97 | 0.97 | 2700 |
| weighted avg | 0.97 | 0.97 | 0.97 | 2700 |

همچنین معیار AUC به صورت وزن دار[15] محاسبه شده است :

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score,roc_curve
roc_auc_score(one_hot_targets.reshape((len(y_test),-
1)), probs, multi_class='ovo', average='weighted')#, labels=['quran' , 'bi
ble'  , 'mizan'])
```

[15] Weighted

0.9982576131687242

و نمودار ROC به صورت زیر می‌باشد :

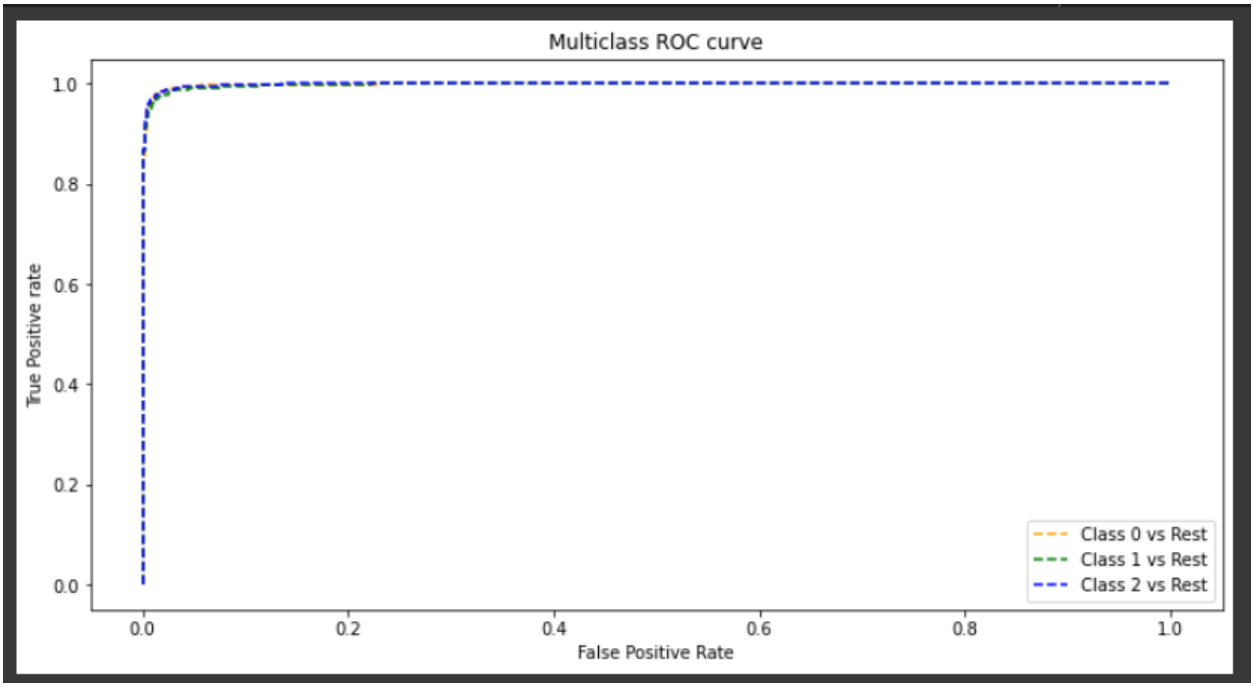


Fig.7 Roc Curve for multi class(Persian set)

## 3. Multi-Lingual-XlmRoberta

فرآیند مشابه قسمت دوم سوال اول را در پیش می‌گیریم.

در ابتدا نوع مدل و Tokenizer را ایجاد می‌کنیم :

```
from transformers import XLMRobertaTokenizer, XLMRobertaForSequenceClassif
ication

MODEL_TYPE = 'xlm-roberta-base'

tokenizer = XLMRobertaTokenizer.from_pretrained(MODEL_TYPE)
```

در زیر پارامترهای مورد نیاز برای آموزش را آورده‌ایم :

```
MODEL_TYPE = 'xlm-roberta-base'


L_RATE = 3e-5
MAX_LEN = 128

NUM_EPOCHS = 3
```

```
BATCH_SIZE = 32
NUM_CORES = os.cpu_count()

NUM_CORES
#torch.cuda.memory_summary(device=None, abbreviated=False)
```

در این قسمت چون یک جمله مبدا و یک جمله مقصد داریم، در تعریف کلاس مربوط به دادگان تست در قسمت getitem، باید جمله دوم هم اضافه کنیم :

```
def __getitem__(self, index):

        # get the sentence from the dataframe
        sentence1 = self.df_data.iloc[index]['sent1']
        sentence2 = self.df_data.iloc[index]['sent2']
```

مشابه قبل، مدل را به کمک sequence classification تعریف می‌کنیم :

```
from transformers import XLMRobertaForSequenceClassification,AdamW

model = XLMRobertaForSequenceClassification.from_pretrained(
    MODEL_TYPE,
    num_labels = 3, # The number of output labels. 2 for binary classifica
tion.
)

# Send the model to the device.
model.to(device)
```

لایه آخر مربوط به Classification در زیر آورده شده است :

```
  (classifier): RobertaClassificationHead(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (out_proj): Linear(in_features=768, out_features=3, bias=True)
  )
)
```

در نهایت نتیجه آموزش مدل در زیر آورده شده است :

```
======== Epoch 1 / 3 ========
Training...
Train loss: 86.8382795038633
```

Validation...
Val loss: 7.309636967023835
Val acc:  0.9803703703703703

======== Epoch 2 / 3 ========
Training...
Train loss: 19.58212022675434

Validation...
Val loss: 4.17649958160473
Val acc:  0.9892592592592593

======== Epoch 3 / 3 ========
Training...
Train loss: 12.658548310020706

Validation...
Val loss: 5.32082277329755
Val acc:  0.9874074074074074

برچسب های پیشبینی شده توسط مدل بر روی دادگان تست به صورت زیر می‌باشد :

```
    # Take the argmax. This returns the column index of the max value in
 each row.

preds = np.argmax(stacked_preds, axis=1)

preds
```

array([2, 1, 1, ..., 0, 2, 1])

در نهایت، گزارش طبقه بندی مدل بر روی دادگان تست آورده شده است :

F1: 0.9881538410205132

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| quran | 0.98 | 0.99 | 0.99 | 900 |
| bible | 1.00 | 0.98 | 0.99 | 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 |

همانطور که قبلا ذکر شد، خروجی های پیشبینی شده توسط مدل برای استفاده در تابع auc، باید به صورت احتمال درآیند.
توسط تابع softmax، به این مهم می‌رسیم :

```
torch.softmax(torch.tensor(stacked_preds),dim=1).numpy()#.sum(dim=1)
```

```
array([[6.2979780e-05, 5.6871071e-05, 9.9988008e-01],
       [1.9856830e-05, 9.9990749e-01, 7.2565279e-05],
       [3.0287349e-05, 9.9989879e-01, 7.0971219e-05],
       ...,
       [9.9994874e-01, 1.2493905e-05, 3.8694183e-05],
       [5.7074121e-05, 5.0351115e-05, 9.9989259e-01],
       [1.2542278e-05, 9.9988186e-01, 1.0561551e-04]], dtype=float32)
```

در نهایت مقدار AUC و همچنین نمودار ROC را رسم می‌کنیم :

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score,roc_curve
roc_auc_score(one_hot_targets.reshape((len(y_test),-
1)), torch.softmax(torch.tensor(stacked_preds),dim=1).numpy(), multi_class
='ovo', average='weighted')#, labels=['quran' , 'bible'  , 'mizan'])
```

0.9989836419753088

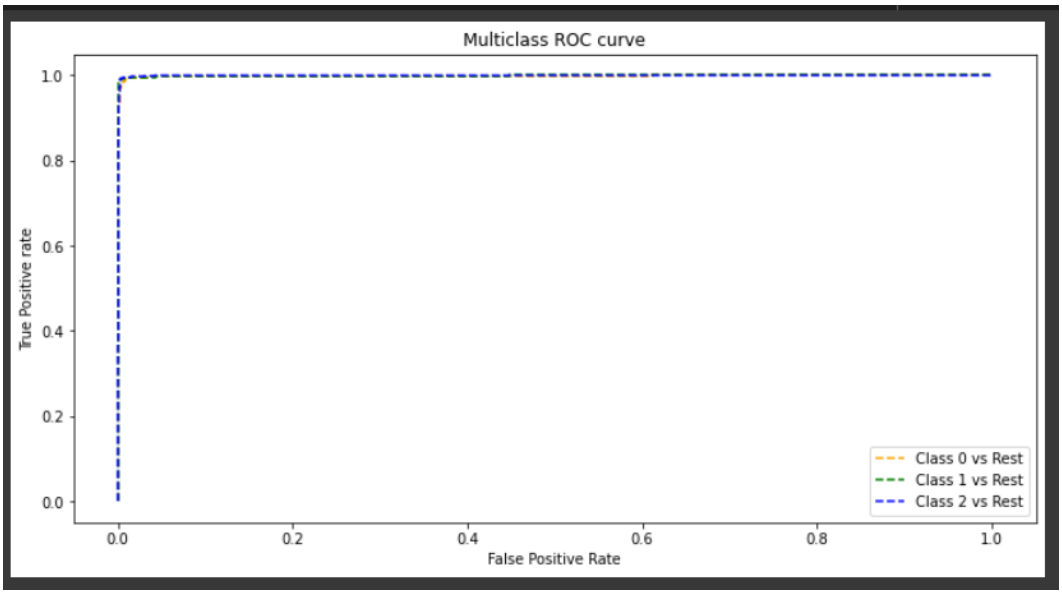


Fig.8 ROC Curve for Multi Class(Persian,English sets)

پس همانطور که مشخص است، عملکرد بر روی مدل چند زبانه بهتر از مدل تک زبانه شد.

# Cross-lingual zero-shot transfer learning(Bonus)

## 1. Expectation

با توجه به اینکه مدل Roberta ، یک مدل Multilingual می‌باشد، پس مدل Pre-trained شده آن باید اطلاعات تجمیعی از Embedding دو فضای مختلف زبان فارسی و انگلیسی داشته باشد. ولی وقتی مدل را Fine-Tune می‌کنیم، مدل بر روی دادگان آموزش که از نوع دادگان انگلیسی هستند احتمالا OverFit می‌شود. پس احتمالا نتیجه‌ی مطلوبی بر روی دادگان تست که از نوع فارسی هستند نگیریم.

## 2. Performance

برای آموزش مدل، تنها تغییری که باید اعمال کرد این است که در تعریف کلاس مربوط به دادگان تست، در تابع getitem، باید جملات مربوط به ستون دوم(جملات فارسی به جای انگلیسی) قرار داد شود :

```
class TestDataset(Dataset):

    def __init__(self, df):
        self.df_data = df



    def __getitem__(self, index):

        # get the sentence from the dataframe
        sentence1 = self.df_data.iloc[index]['targets']
        #sentence2 = self.df_data.iloc[index]['targets']

        # Process the sentence
        # --------------------
```

حال مدل را به تعداد Epoch 10 آموزش می‌دهیم :

```
======== Epoch 1 / 10 ========
Training...
Train loss: 98.85888106003404

Validation...
Val loss: 10.139117512851954
Val acc:  0.965925925925926

======== Epoch 2 / 10 ========
Training...
Train loss: 42.38385655230377
```

Validation...
Val loss: 11.8182415420888
Val acc:  0.9688888888888889

======== Epoch 3 / 10 ========
Training...
Train loss: 25.063166364008794

Validation...
Val loss: 12.028178311273223
Val acc:  0.9722222222222222

======== Epoch 4 / 10 ========
Training...
Train loss: 14.305375450552674

Validation...
Val loss: 9.526650888859876
Val acc:  0.9796296296296296

======== Epoch 5 / 10 ========
Training...
Train loss: 11.840638816560386

Validation...
Val loss: 11.591414684306073
Val acc:  0.9766666666666667

======== Epoch 6 / 10 ========
Training...
Train loss: 9.916480610714643

Validation...
Val loss: 11.526229306939058
Val acc:  0.9796296296296296

======== Epoch 7 / 10 ========
Training...
Train loss: 7.969135289971746

Validation...

Val loss: 10.609145741727843
Val acc:  0.9796296296296296

======== Epoch 8 / 10 ========
Training...
Train loss: 5.333078030478646

Validation...
Val loss: 9.974094127726858
Val acc:  0.9851851851851852

======== Epoch 9 / 10 ========
Training...
Train loss: 7.996978300197952

Validation...
Val loss: 12.837541228349437
Val acc:  0.9807407407407407

======== Epoch 10 / 10 ========
Training...
Train loss: 9.858441431540996

Validation...
Val loss: 13.525319684882561
Val acc:  0.9803703703703703

و گزارش طبقه بند را بر روی دادگان تست بدست می‌آوریم :

F1: 0.583482510821733

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| quran | 0.85 | 0.57 | 0.68 | 900 |
| bible | 0.78 | 0.27 | 0.40 | 900 |
| mizan | 0.50 | 1.00 | 0.67 | 900 |
| accuracy | | | 0.61 | 2700 |
| macro avg | 0.71 | 0.61 | 0.58 | 2700 |
| weighted avg | 0.71 | 0.61 | 0.58 | 2700 |

طبق نتایج بدست آمده، می‌توان گفت که تا حدی پیشبینی قسمت اول درست بود ولی یک نکته مهم وجود دارد.
*اطلاعات دقت در دادگان ارزیابی در حین آموزش مدل اطلاعاتی به ما نمی‌دهد و حتی در صورتی که دقت در دادگان ارزیابی رو به افزایش باشد، نمی‌توان نظری در مورد دادگان تست داد چون نوع دادگان این دو با هم متفاوت است.
در واقع اگر مدل برای تعداد Epoch های کمتری آزمایش می‌کردیم، شاید دقت بر روی دادگان تست کمی بهتر هم می‌شد.
نکته مهم دیگر در عملکرد بدست آمده در هر کلاس می‌باشد که عملکرد دو کلاس قرآن و شاهکار ادبی تقریبا یکی می‌باشد ولی بر روی کتاب انجیل ضعیف می‌باشد ، این نشان می‌دهد که مدل بر روی دادگان با برچسب bible بیش از حد Over Fit شده و Embedding مربوط به دادگان فارسی در این دسته توسط لایه Feedforward بسیار کمتر اثر داده شده است.

## 3. Applications

در هنگامی که تعداد داده زیادی برای آموزش مدل بر روی یک زبان خاص نداریم، یک مدل جامع که بر روی چندین زبان مختلف آموزش دیده را لود می‌کنیم و آنرا بر روی دادگان انگلیسی Fine-Tune می‌کنیم. در نهایت بر روی دادگان تست پیشبینی انجام می‌دهیم.

برای مثال برای در شبکه های اجتماعی مثل Twitter که User هایی با زبان های مختلف وجود دارد و هر کشور به خاطر محدودیت منابع نمی‌تواند صرفا برای ملیت خود یک مدل آموزش دهد. برای همین از یک مدل جامع تر که بر روی چندین زبان مختلف آموزش دیده استفاده می‌کند.