به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین پنجم

نام و نام خانوادگی	فاطمه جلیلی - سالار صفردوست
شماره دانشجویی	۸۱۰۱۹۹۳۹۸ - ۸۱۰۱۹۹۴۵۰
تاریخ ارسال گزارش	۱۴۰۲/۱۰/۱۸

فهرست

پاسخ ۱. تشخیص احساسات گفتار(SER)

۱-۲-۱. چالش‌های داده‌های صوتی در یادگیری

به طور کلی یادگیری داده‌های صوتی از چند لحاظ از یادگیری کلمات و نوشته پیچیده‌تر می‌باشد، به عنوان مثال:

۱- صوت پیوسته است و اطلاعات در داخل آن به گسستگی اطلاعات داخل کلمات یک جمله نمی‌باشد.

۲- وجود چندین صدای مختلف به طور همزمان داخل قطعه‌ی صوتی، باعث می‌شود کار تشخیص و جدا کردن اطلاعات مهم سخت باشد. (گرفتن instance دشوار است.)

۳- داخل صدا هیچ الفبای مشخصی برای استفاده در واحدهای صوتی وجود ندارد و این الفبا باید استخراج شود. (بر خلاف نوشته‌ها)

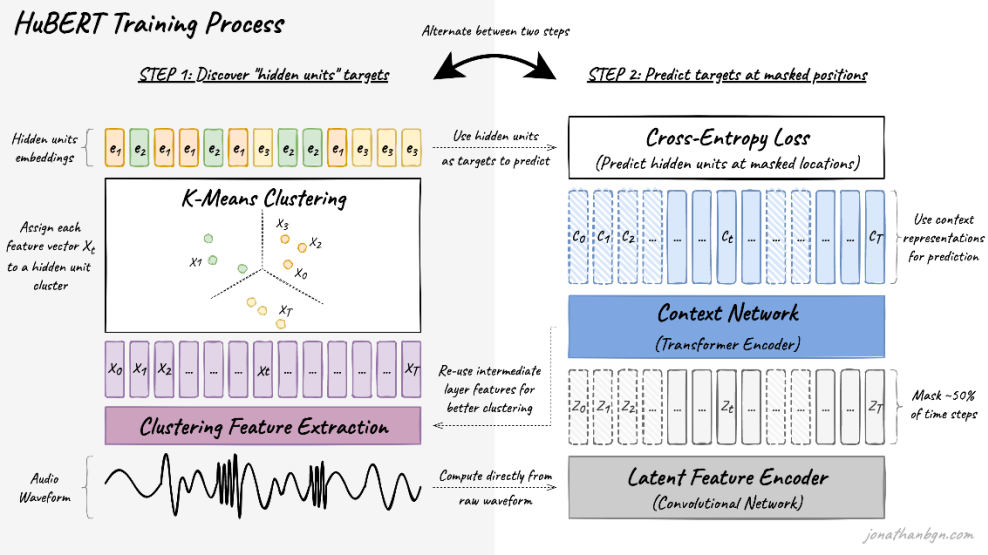
۴- هیچ علامت مشخصی برای نشان دادن اتمام یک فریم آوایی وجود ندارد. (برخلاف وجود فاصله در نوشته)

۱-۲-۲. رویکرد HuBert

نویسندگان مقاله، برای رفع مشکلات مطرح شده روشی را پیش می‌گیرند که برای آموزش مدل نیازی به لیبل‌های واقعی نباشد.

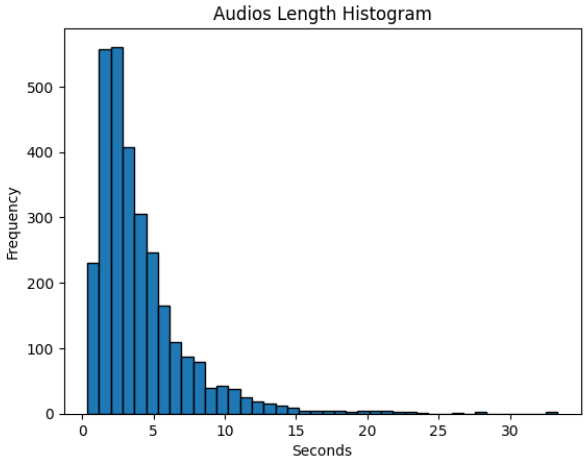
در این روش در واقع مدل یاد می‌گیرد تا رفتار خود را به سمت یک مدل کلاسترینگ پیش ببرد، به این شکل که در هر مرحله داده‌ی صوتی از داخل یک Acoustic Unit Discovery System عبور می‌کند که مطابق مقاله با استفاده از الگوریتم k-means روی تبدیل MFCC صورت می‌گیرد و سپس خروجی این واحد به عنوان لیبل برای مدل HuBert عمل می‌کند تا این مدل بتواند یاد بگیرد از داخل یک داده‌ی صوتی اطلاعات مهم آن را دسته بندی کند.

مدل Bert در واقع داخل مدل Hubert استفاده شده است و چگونگی استفاده از آن به این صورت است که اینبار به جای ورودی‌های وکتوری کلمه، در یک مرحله feature extraction کانولوشنالی صوت را به صورت دنباله‌ی وکتوری در می‌آوریم و سپس این دنباله را به Bert می‌دهیم. مدل Bert هم به همان شیوه‌ی masking آموزش می‌بیند تا بتواند وکتور ورودی را در صورت نبود چند ورودی از آن(ماسک شده) همچنان بازسازی کند و پیش‌بینی کند.

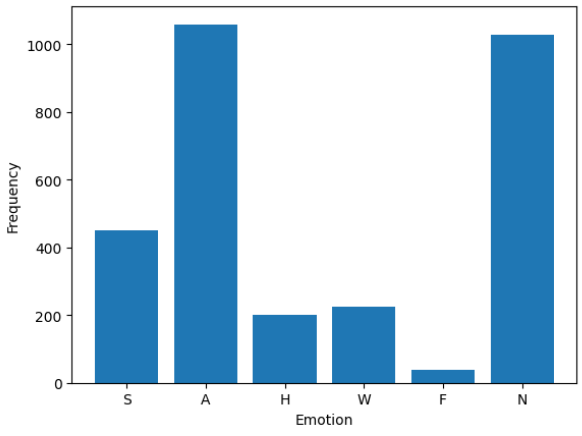


خلاصه‌ای از آموزش مدل **Hubert**

۱-۳-۱. پیش‌پردازش داده‌ها



هیستوگرام طول سیگنال‌های ورودی بر حسب ثانیه



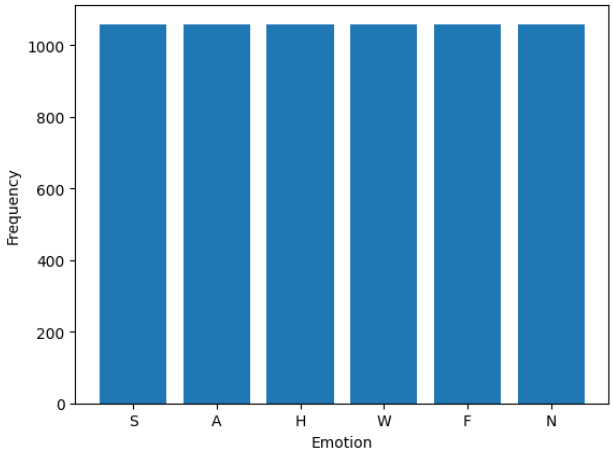
توزیع داده‌های هر کلاس قبل از آگمنت کردن

برای قسمت پیش‌پردازش دو عملیات صورت گرفت:

۱- در هنگام خواندن داده‌ها فرکانس نمونه برداری تمامی سیگنال‌ها به ۱۶ کیلوهرتز برده شد. (فرکانسی که مدل روی آن پری ترین شده است.)

۲- توزیع داده‌های کلاس‌ها بالانس نبوده و برای انجام این کار در یک تابع اعضای هر کلاس را با استفاده از آگمنت کردن صدا توسط (تغییر پیچ صدا و طول آن) به مقدار کلاس با عضو بیشتر رساندیم.

برای دو کار بالا از کتابخانه‌ی liberosa استفاده شد.



توزیع داده‌های هر کلاس پس از آگمنت کردن

در نهایت یک تنسور به اسم audios_indices به طول مجموع سیگنال‌ها ساخته شد تا انجام کار دیتالودینگ به علت تفاوت طول سیگنال‌ها پیچیدگی نداشته باشد و کار تقسیم بندی داده‌ها بر روی این indexها انجام شود نه بر روی لیست صداها.

```
[12] audio_indices = torch.tensor(range(len(audio_waves)))
     audio_labels = torch.tensor(audio_labels)

[13] seed = 10
     train_indices, test_indices, train_labels, test_labels = train_test_split(audio_indices, audio_labels, test_size=0.2, random_state=seed)
     test_indices, eval_indices, test_labels, eval_labels = train_test_split(test_indices, test_labels, test_size=0.5, random_state=seed)

[14] print(train_indices.shape)
     print(test_indices.shape)
     print(eval_indices.shape)

     torch.Size([5083])
     torch.Size([635])
     torch.Size([636])
```

تقسیم‌بندی داده‌ها به سه مجموعه

۱-۳-۲. ساخت دیتالودر

برای ساخت دیتالودر از دیتالودر دیفالت استفاده شد. در واقع اگر می‌خواستیم دیتالودر را روی لیست صداها پیاده‌سازی کنیم به پیچیدگی‌هایی برخورد می‌کردیم که با استفاده از ایندکس صداها به جای خود صداها از این پیچیدگی جلوگیری کردیم.

```
feature_extractor = Wav2Vec2FeatureExtractor.from_pretrained('facebook/wav2vec2-base-960h')

[28] dataset = TensorDataset(train_indices, train_labels)

[29] batch_size = 2
     dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

[ ]  # for indices, label in dataloader:
     #   x = [audio_waves[i].tolist() for i in indices]
     #   x = feature_extractor(x, sampling_rate=target_fs, padding=True, return_tensors='pt')
```

کد مربوط به ساخت دیتالودر

مراحل پدینگ صدا در هر batch در داخل این دیتالودر تعبیه نشد، به جای آن در حین آموزش مدل اینکار صورت می‌گیرد که همانطور که گفته شد اینکار برای جلوگیری از پیچیده شدن دیتالودر بود.

پدینگ در هر batch با پدینگ بر روی کل داده‌ها به شیوه‌ی max_length مزایا و معایبی دارد:

۱- پدینگ روی تمام داده‌ها امکان استفاده‌ی راحتتر از کل داده را به ما می‌دهد، چرا که طول تمامی سیگنال‌ها یکی شده است.

۲- پدینگ روی تمام داده‌ها حجم داده‌ها را بسیار زیاد می‌کند، کافیست یک داده‌ی با طول زیاد و پرت داشته باشیم تا حجم داده‌ها چند برابر بشود.

۳- آموزش مدل روی سیگنال‌های با طول یکسان ممکن است موجب ایجاد بایاس در پیش‌بینی مدل شود.

در نتیجه‌ی این گزینه‌ها از روش پدینگ بر روی هر batch استفاده کردیم.

برای انجام اینکار و نرمال کردن داده‌ها از FeatureExtractor موجود در مدل Wav2Vec2 استفاده شد.

۱-۴-۱. تولید بازنمایی مناسب از کل ورودی

خروجی لایه‌ی آخر مدل Hubert دنباله‌ای از وکتورها می‌باشد و اطلاعات بیش از اندازه‌ای دارد که برای کلسیفیکیشن لازم نیست. برای آنکه این داده‌ها را خلاصه کنیم، یک بردار representation را در خروجی مدل به گونه‌ای تعریف می‌کنیم که میانگین این وکتورها را در راستای زمانی بگیرد. (روش‌های دیگری مانند پیدا کردن ماکسیمم نیز از کارهایی است که می‌توانستیم انجام دهیم.)

سپس این بردار را به شبکه‌های FC بعدی می‌دهیم تا کار کلسیفیکیشن را انجام دهند.

این عملیات در مابین تعریف مدل به اسم Pool قرار داده شده است.

۱-۴-۲. آموزش مدل

```
[32] class EmotionClassifier(nn.Module):
       def __init__(self, Hubert, num_classes):
         super(EmotionClassifier, self).__init__()
         self.Hubert = Hubert
         self.Pool = torch.mean
         self.FC1 = nn.Linear(Hubert.config.hidden_size, num_classes)
         self.FC2 = nn.Linear(num_classes, num_classes)
       def forward(self, inputs):
         x = self.Hubert(inputs)
         representation_vector = self.Pool(x.last_hidden_state, dim=1)
         x = self.FC1(representation_vector)
         x = self.FC2(x)
         return x
```

مدل تعریف شده

* همانگونه که در قسمت دیتالودر گفته شد عملیات پدینگ در هر batch در قسمت آموزش با فراخوانی FeatureExtractor انجام می‌شود.

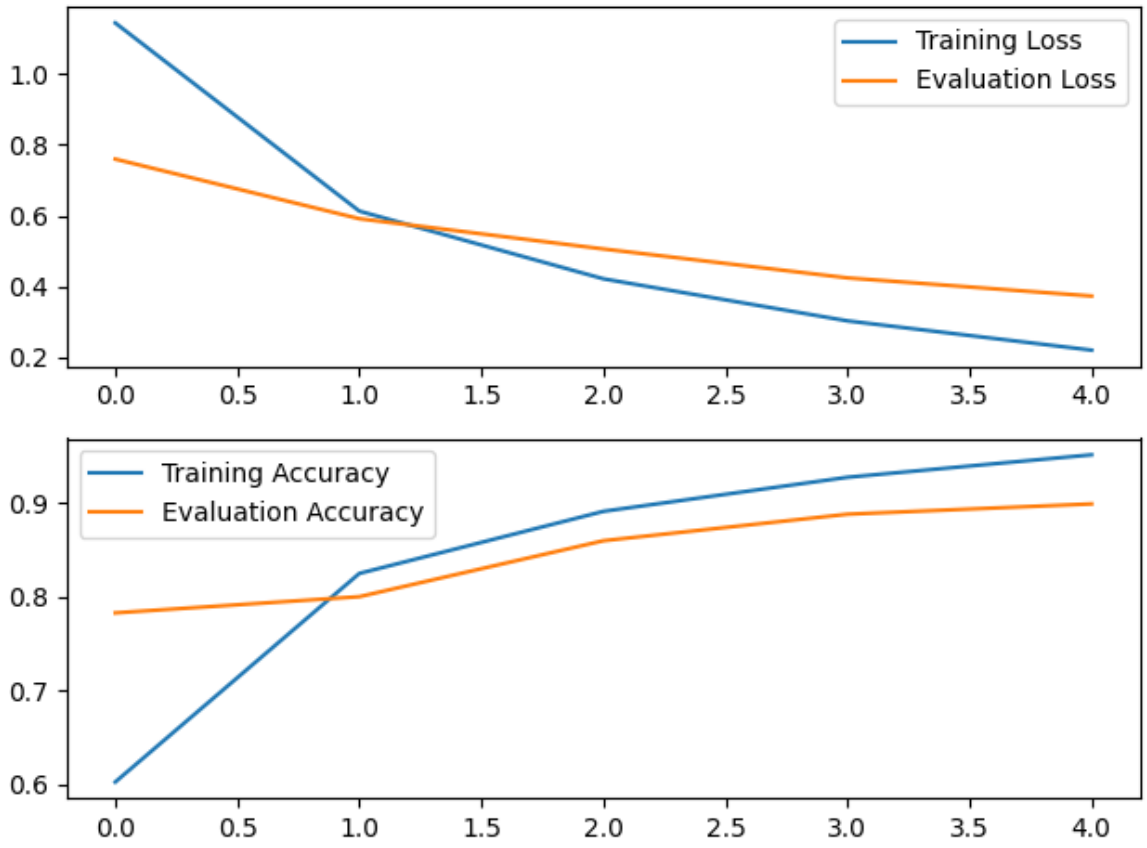
پارامترهای تنظیم شده برای آموزش مدل به شکل زیر است:

Batch size = 2

Epochs = 5

Learning rate = $5*10^{-6}$

Criterion = CrossEntropyLoss



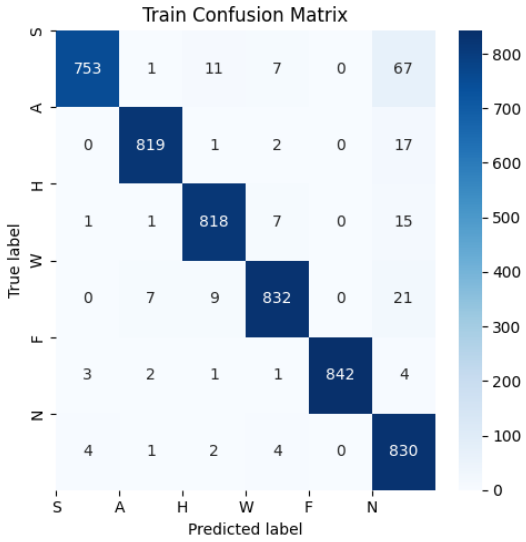
نمودارهای به دست آمده حین آموزش روی کل داده‌های آموزش و ارزیابی در هر epoch

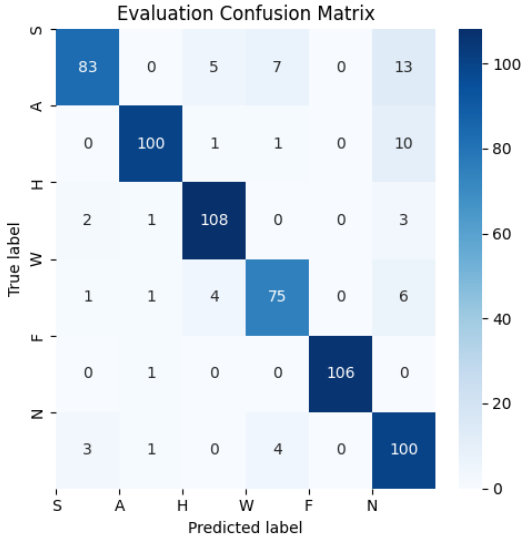
```
print(f'Train Accuracy: {train_accuracy}')
print(f'Eval Accuracy: {eval_accuracy}')
print(f'Test Accuracy: {test_accuracy}')
```

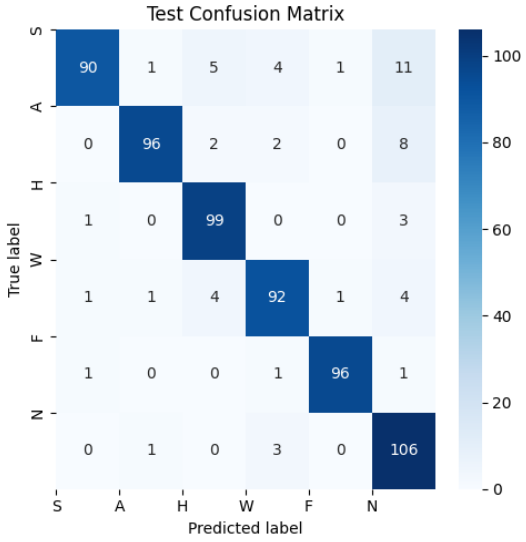
```
Train Accuracy: 0.9628172339169782
Eval Accuracy: 0.89937106918239
Test Accuracy: 0.9118110236220472
```

دقت مدل روی مجموعه‌های دیتاست

در صفحه‌ی بعد نیز ماتریس درهم ریختگی هر سه مجموع رسم شده است. تمامی نتایج حاکی از همگرایی خوب و عدم وجود بایاس و واریانس در فرآیند آموزش مدل می‌باشد. همانگونه که دیده می‌شود تقریباً در تمامی کلاس‌ها مشکلی بابت تشخیص نداشته‌ایم و کمترین دقت تشخیص برای کلاس‌های سورپرایز شدن و ناراحتی بوده است.







ماتریس‌های در هم ریختگی هر سه مجموعه

پاسخ ۲. تنظیم دقیق مدل BERT

. پیش‌پردازش داده‌ها

داده های آموزش (**train**) :

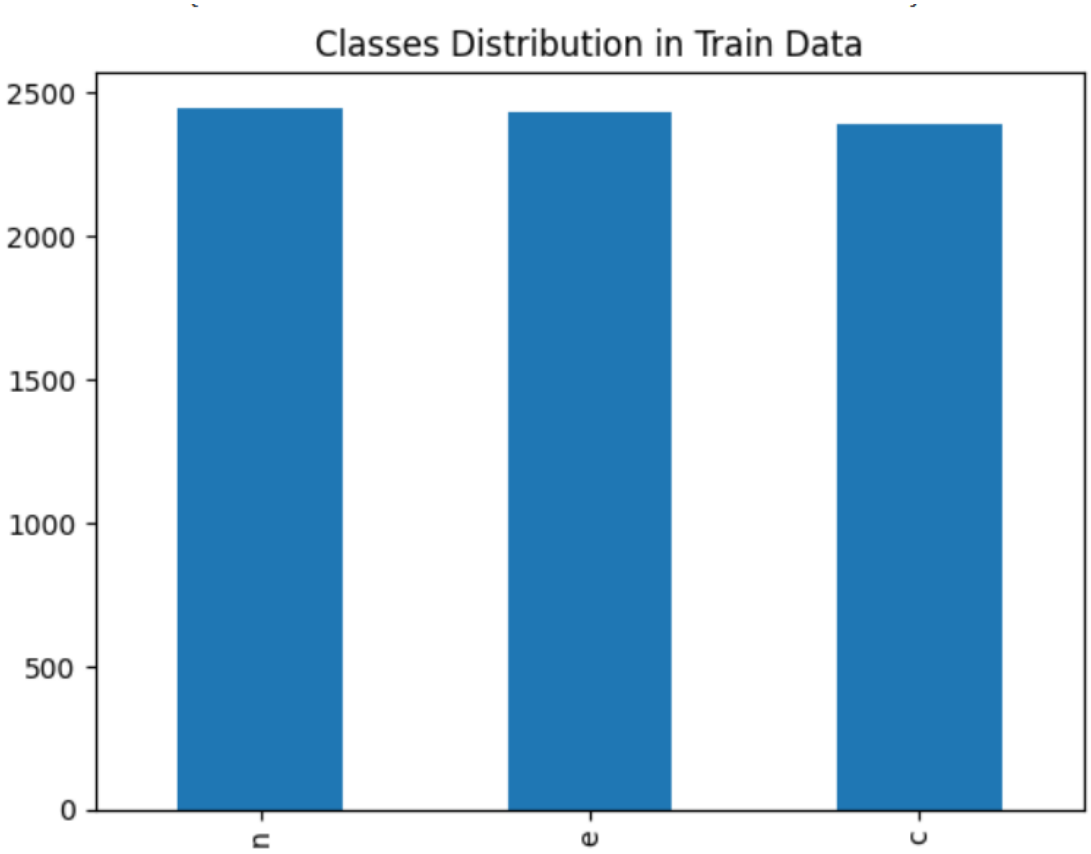
تعدادی از نمونه ها :

	premise	hypothesis	label
0	... اولین انتقال و نفوذ طبیعی فرهنگ و تمدن اسلامی	نخستین انتقال و نفوذ طبیعی فرهنگ و تمدن اسلامی...	e
1	... اولین انتقال و نفوذ طبیعی فرهنگ و تمدن اسلامی	کانون های جغرافیایی مصر، اندلس و شام، نخستین ر...	c
2	... اولین انتقال و نفوذ طبیعی فرهنگ و تمدن اسلامی	سیسیل بعد از اسپانیا بزرگ ترین کانونی بود که ه...	n
3	ویژگی های هنر عصر اموی: ۱- تلفیقی بودن ۲- بازن...	نقاشی های تزئینی و تندیس های بی‌کیفیت، یکی از ...	e
4	ویژگی های هنر عصر اموی: ۱- تلفیقی بودن ۲- بازن...	با کیفیت بودن تندیس های دوره اموی، یکی از ویژگ...	c

تعداد نمونه ها که مطابق شکل مشاهده می کنیم ۷۲۶۶ نمونه است .

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7266 entries, 0 to 7265
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   premise     7266 non-null   object
 1   hypothesis  7266 non-null   object
 2   label       7266 non-null   object
```

توزیع نمونه های در دسته های **label** :



مطابق فوق مشاهده می کنیم تقریبا به تعداد برابر از هر کلاس در نمونه ها وجود دارد و دسته **c** در مقایسه با بقیه تعداد کم تری نمونه دارد

label	premise	hypothesis
c	2389	2389
e	2429	2429
n	2448	2448

داده های تست (**test**) :

تعدادی از نمونه ها:

	premise	hypothesis	label	hard(hypothesis)	hard(overlap)
0	دوران امامت امام صادق علیه السلام، مصادف است ب...	امام سجاد (ع) در دورانی امامت کردند که همزمان ...	c	0	1
1	دوران امامت امام صادق علیه السلام، مصادف است ب...	دستگاه فاسد حکومتی با صرف هزینه های هنگفت، سعی...	n	1	0
2	با شهادت امام رضا(ع) مرحله جدیدی از تلاش ائمه ...	دوران محنت اهل بیت پس از شهادت امام رضا(ع) آغا...	e	0	0
3	با شهادت امام رضا(ع) مرحله جدیدی از تلاش ائمه ...	بعد از به شهادت رسیدن امام هادی(ع) دوران محنت ...	c	1	1
4	با شهادت امام رضا(ع) مرحله جدیدی از تلاش ائمه ...	حضرت جواد(ع) در سال ۱۹۵ هجری در مدینه ولادت یافت.	n	1	0

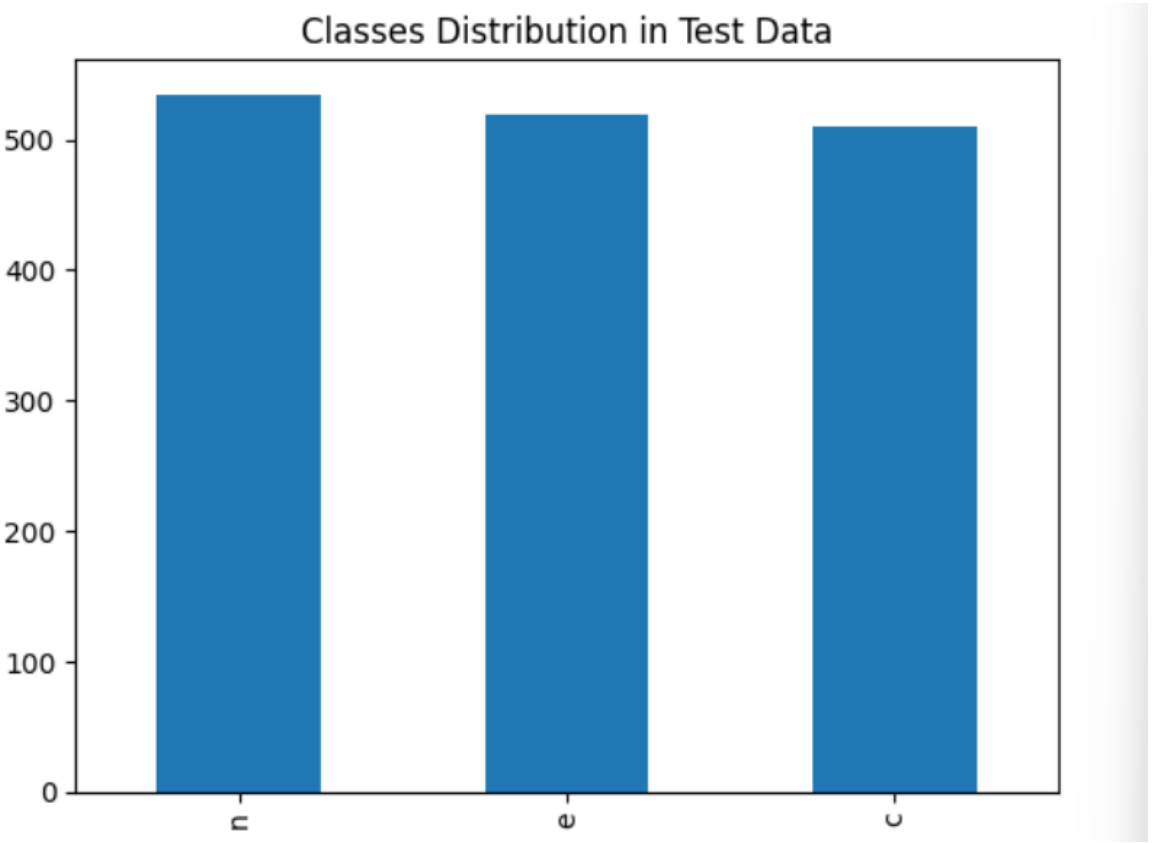
در اولین گام پیش پردازش ستون های آخر که مربوط به **hard(hypothesis)** و **hard(overlap)** هستند را در داده تست حذف می کنیم چرا که نیازی به آنها نداریم.

تعداد نمونه ها که مطابق شکل مشاهده می کنیم ۱۵۴۶ نمونه است .

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1564 entries, 0 to 1563
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   premise     1564 non-null   object
 1   hypothesis  1564 non-null   object
 2   label       1564 non-null   object
```

توزیع نمونه ها در دسته های **label** :

مشاهده می کنیم تقریبا به تعداد برابر از هر کلاس در نمونه ها وجود دارد و دسته **c** در مقایسه با بقیه تعداد کم تری نمونه دارد.



label	premise	hypothesis
c	510	510
e	519	519
n	535	535

داده های ارزیابی (**validation**):

تعدادی نمونه :

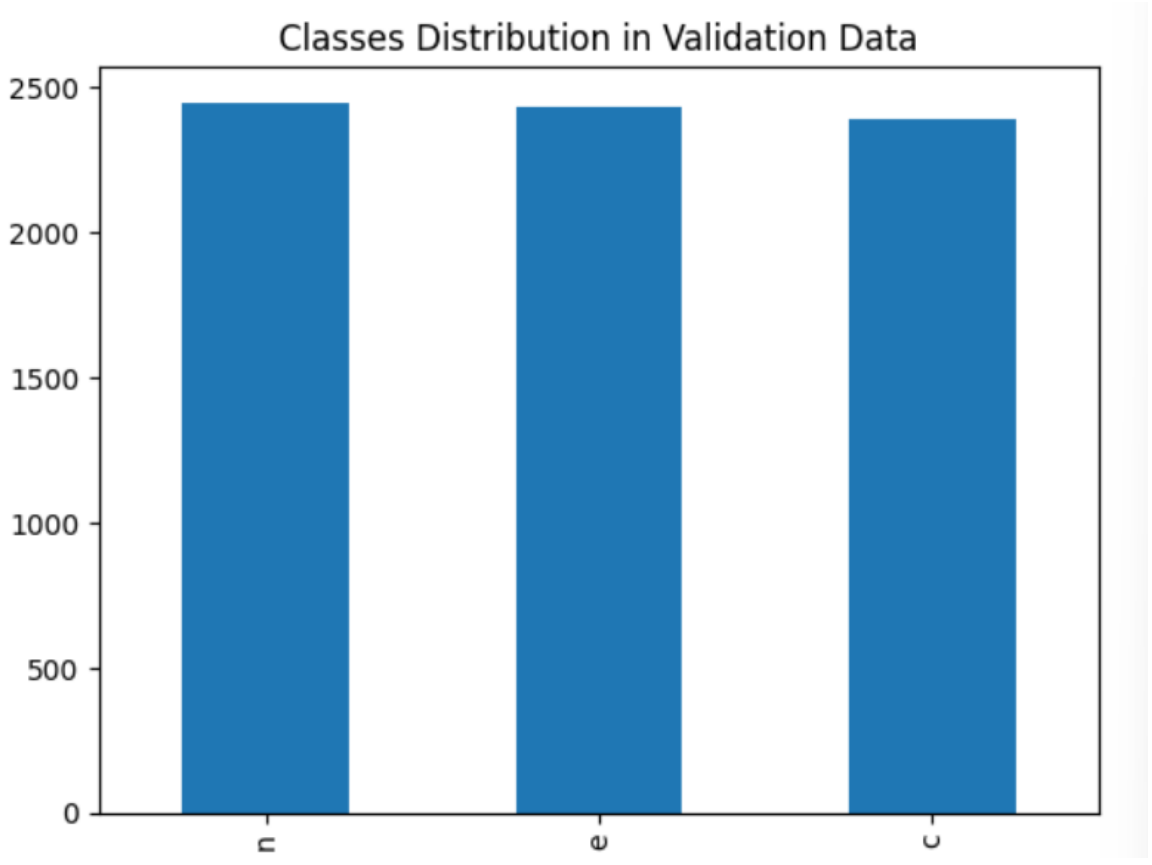
	premise	hypothesis	label
0	یکی از سرآمدانی که بر تارک علمی مدرسه اسکندریه...	...کتاب مگیستی (المجسطی) از آثار بطلمیوس است و از	e
1	یکی از سرآمدانی که بر تارک علمی مدرسه اسکندریه...	المجسطی (مگیستی)، یکی از کتاب های هرون از مکتب...	c
2	یکی از سرآمدانی که بر تارک علمی مدرسه اسکندریه...	... ترجمه المجسطی، پایه تهیه جداول نجومی گردید که	n
3	تاریخ پزشکی یونانی با بقراط معروف است. آثار طب...	.کتاب کلمات قصار یا فصول، اثر بقراط است	e
4	تاریخ پزشکی یونانی با بقراط معروف است. آثار طب...	.جالینوس، نویسنده کتاب کلمات قصار بوده است	c

تعداد نمونه ها که مطابق شکل مشاهده می کنیم ۱۵۳۷ نمونه است .

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1537 entries, 0 to 1536
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   premise     1537 non-null   object
 1   hypothesis  1537 non-null   object
 2   label       1537 non-null   object
```

توزیع نمونه ها در دسته های **label** :

label	premise	hypothesis
c	499	499
e	515	515
n	523	523



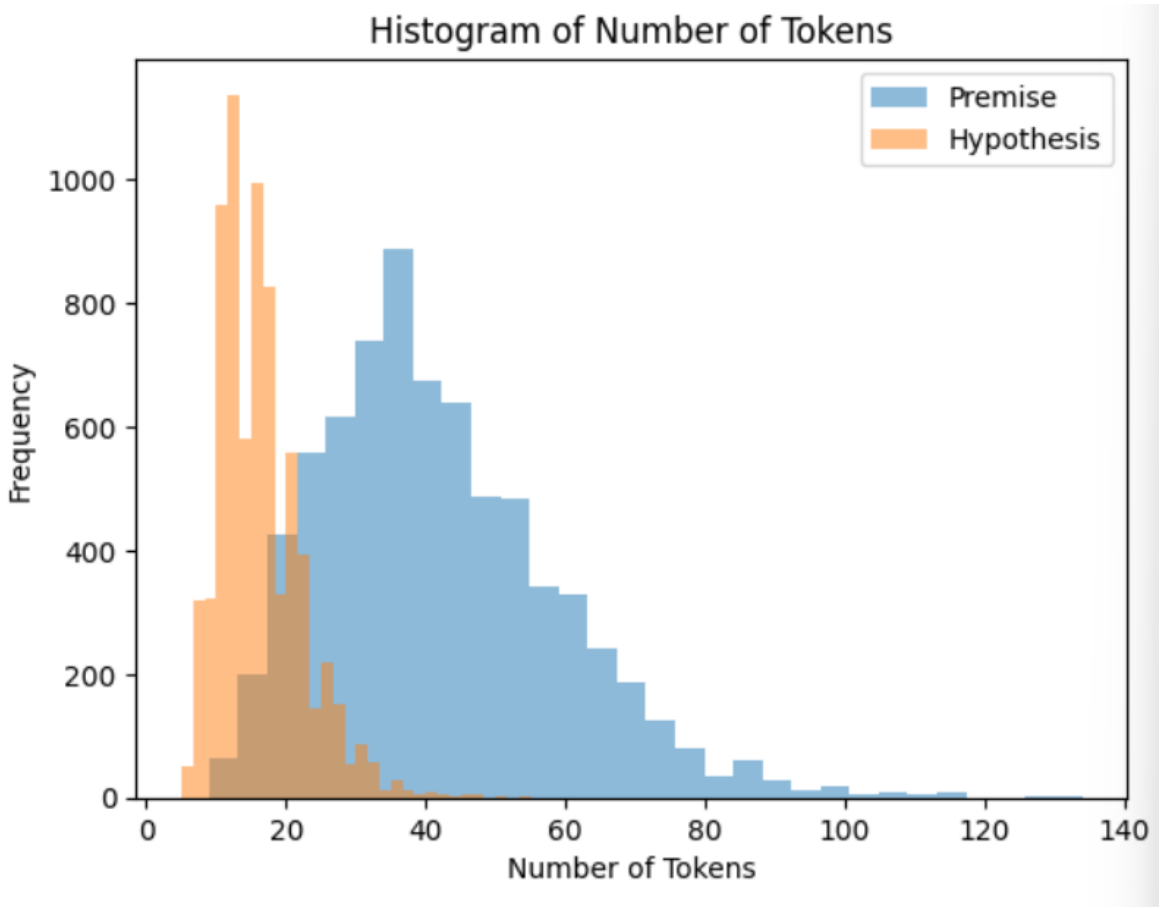
مطابق دیگر داده ها در اینجا هم تعداد نمونه های منعلق به دسته **c** کمی کم تر است.

در گام بعدی پیش پردازش **label** های **n,e,c** را با اعداد ۰و ۱و ۲ جایگزین می کنیم .

در گام بعدی هم پس از لود کردن **tokenizer** مربوط به مدل معرفی شده ماکسیمم تعداد توکن های موجود در جملات داده های **premise** و **hypothesis** را مطابق شکل زیر پرینت می کنیم تا در هنگام توکنایز کردن داده های بتوانیم حدود حداکثر طول برای **padding** را بدست آوریم :

```
Maximum number of tokens in premise: 134
Maximum number of tokens in hypothesis: 55
```

مطابق آنچه خواسته شده تعداد توکن ها در جملات **premise** و **hypothesis** در داده های ترین هم مطابق شکل زیر رسم می کنیم:



در گام آخر با در دست داشتن حداکثر طول حدود حاصل جمع دو ماکسیمم طول بدتس آمده فوق مطابق آنچه توضیح داده شد با استفاده از کتابخانه پاندا دیتاست ها ورودی مدل را با توکنایز کردن آماده می کنیم:

```
def batchTokenize(sample):
    return tokenizer(sample["premise"], sample["hypothesis"],  padding='max_length', truncation=True, max_length = 200)

train_dataset = Dataset.from_pandas(train_df)
val_dataset = Dataset.from_pandas(val_df)
test_dataset = Dataset.from_pandas(test_df)

train_dataset = train_dataset.map(batchTokenize, batched=True)
val_dataset = val_dataset.map(batchTokenize, batched=True)
test_dataset = test_dataset.map(batchTokenize, batched=True)
```

با توجه به تسک که نیازمند یافتن رابطه منطقی بین **premise** و **hypothesis** است از پیش پردازش های دیگر نظیر ریشه یابی و ... جلوگیری کردیم چرا که به نظر باعث می شوند جملات تغییر زیادی بکنند و تشخیص رابطه ی بین آن ها سخت تر شود.

۲-۱-۲. تنظیم دقیق مدل(fine tune)

برای آشنایی با ساختار مدل آن را پرینت می کنیم تا نام و تعداد لایه های آن را بیابیم :

```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(100000, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=3, bias=True)
)
```

همچنین لایه های مدل را به طور مجزا برای دیدن نام دقیق آن ها پرینت می کنیم (برای نمونه لایه های اولین **berlayer** از میان ۱۲ تا پرینت شده است):

```
*** Embedding Layer ***

bert.embeddings.word_embeddings.weight                  (100000, 768)
bert.embeddings.position_embeddings.weight                (512, 768)
bert.embeddings.token_type_embeddings.weight                (2, 768)
bert.embeddings.LayerNorm.weight                            (768,)
bert.embeddings.LayerNorm.bias                              (768,)

*** First Transformer Out of 12 ***

bert.encoder.layer.0.attention.self.query.weight          (768, 768)
bert.encoder.layer.0.attention.self.query.bias              (768,)
bert.encoder.layer.0.attention.self.key.weight            (768, 768)
bert.encoder.layer.0.attention.self.key.bias                (768,)
bert.encoder.layer.0.attention.self.value.weight          (768, 768)
bert.encoder.layer.0.attention.self.value.bias              (768,)
bert.encoder.layer.0.attention.output.dense.weight        (768, 768)
bert.encoder.layer.0.attention.output.dense.bias            (768,)
bert.encoder.layer.0.attention.output.LayerNorm.weight      (768,)
bert.encoder.layer.0.attention.output.LayerNorm.bias        (768,)
bert.encoder.layer.0.intermediate.dense.weight           (3072, 768)
bert.encoder.layer.0.intermediate.dense.bias               (3072,)
bert.encoder.layer.0.output.dense.weight                 (768, 3072)
bert.encoder.layer.0.output.dense.bias                      (768,)
bert.encoder.layer.0.output.LayerNorm.weight                (768,)
bert.encoder.layer.0.output.LayerNorm.bias                  (768,)

*** Output Layer ***

classifier.weight                                           (3, 768)
classifier.bias                                                (3,)
```

- برای آموزش مدل از ابزار **Trainer** استفاده می کنیم.
- مطابق مقاله **batch_size** را ۱۶ تنظیم می کنیم.
- تعداد **epoch** را ۳ قرار می دهیم
- **Learning_rate** های بین **1e-5** تا **5e-5** مطابق مقاله امتحان شد و نتیجه چندان متفاوت نبود ، روی **3e-5** تنظیم کردیم.

نتایج :

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	0.529611	0.787899
2	0.647900	0.549396	0.811321
3	0.318700	0.738854	0.811971

نتایج روی داده تست :

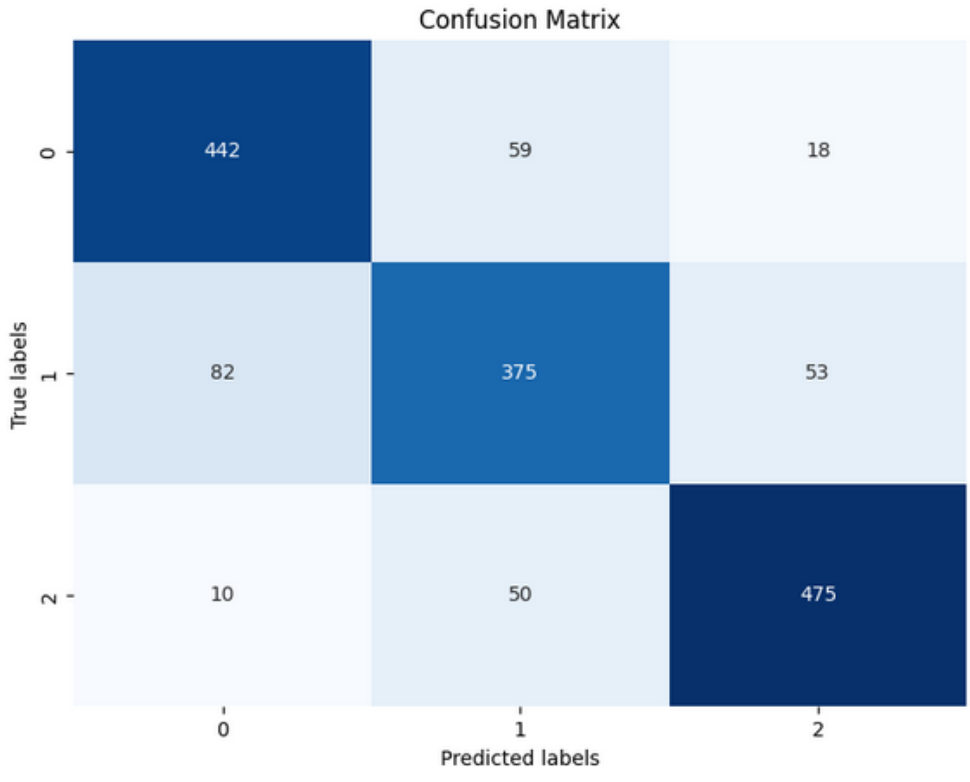
```
[115] trainer.evaluate(test_dataset)
```

```
[98/9
/usr/local/lib/python3.10/dist-packages/
You can avoid this message in future by
Passing `trust_remote_code=True` will be
  warnings.warn(
{'eval_loss': 0.694365918636322,
 'eval_accuracy': 0.8260869565217391,
```

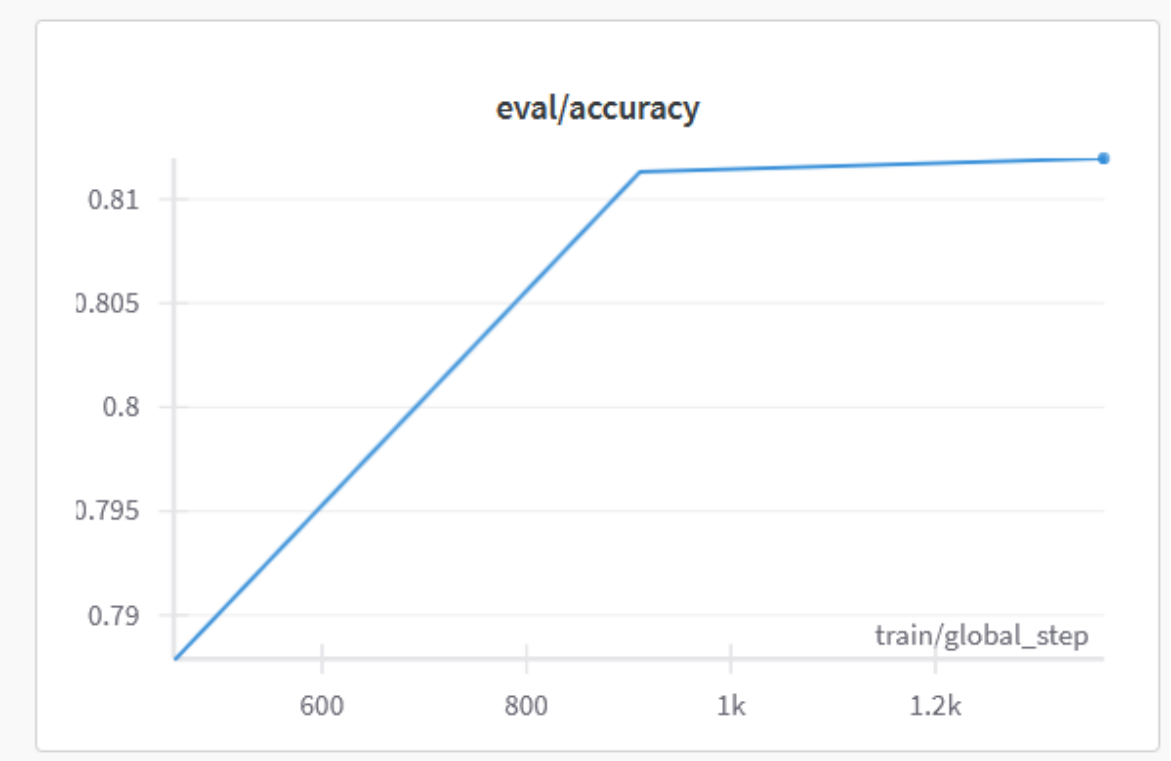
معیار های خواسته شده :

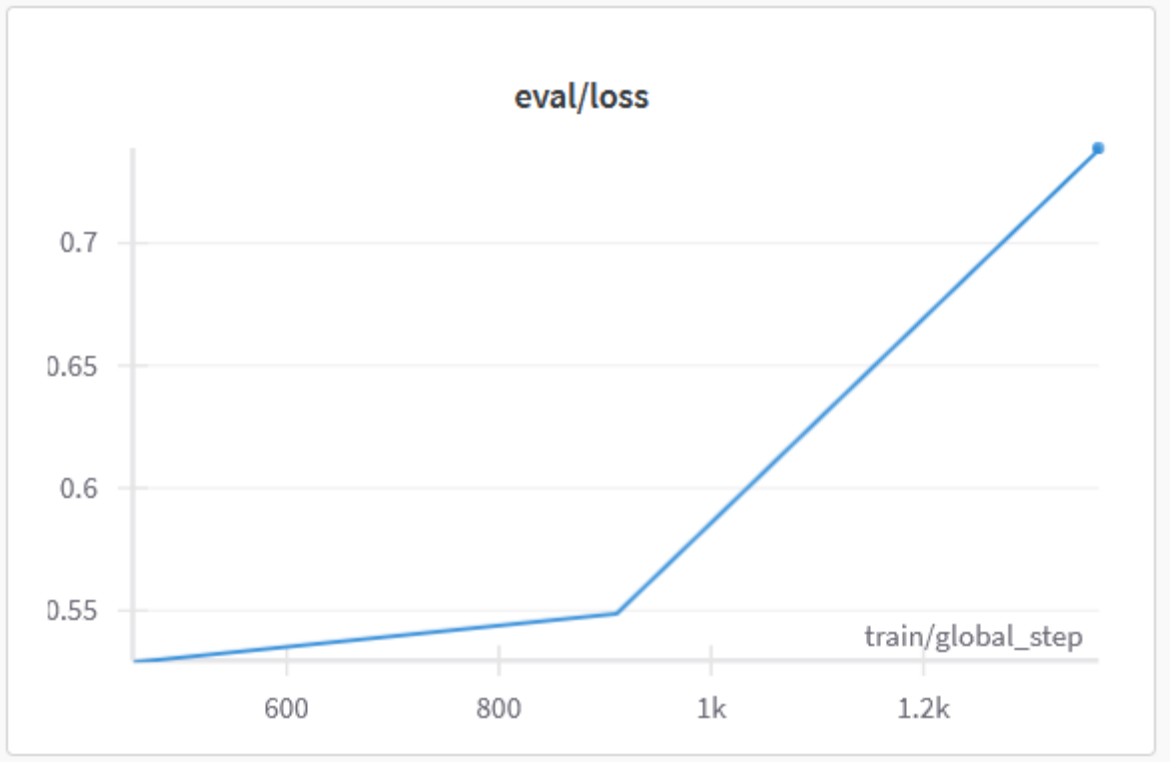
```
              precision    recall  f1-score   support

           0       0.83      0.85      0.84       519
           1       0.77      0.74      0.75       510
           2       0.87      0.89      0.88       535

    accuracy                           0.83      1564
   macro avg       0.82      0.82      0.82      1564
weighted avg       0.82      0.83      0.83      1564
```

Confusion matrix :



نمودار های دقت و خطا روی داده ارزشیابی :





- برای رسم نمودار ها از ابزار **wandb** استفاده کردم که تمام نمودار های مربوطه را همزمان با ترین شدن رسم می کند ، عکس نتایج در گزارش آمده است ، برای دیدن پلات ها در کد آپلود شده نیاز به دسترسی دارین ، من لینک دعوت را به ایمیل شما پیش تر ارسال کرده ام با وارد شدن از طریق آن لینک می توانید پلات ها را در نوت بوک و در سایت **wandb** که لینک مربوط به هر پلات زیر آن قرار دارد مشاهده کنید. نام تیم که لینک دعوت آن برای شما ارسال شده است **fatemeh_nndl** است.

۲-۱-۳. فریز کردن لایه‌های مدل

فریز کردن ۹ لایه ابتدایی :

۹ لایه ابتدایی را مطابق زیر فریز می کنیم و مجدد مشابه قبل نمودار ، معیار ها و ... خواسته شده را رسم می کنیم

```
[ ]  modules = [*model.bert.encoder.layer[:9]]
     for module in modules:
         for param in module.parameters():
             param.requires_grad = False
```

نتایج :

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	0.659488	0.711126
2	0.816600	0.584891	0.768380
3	0.528200	0.591800	0.783995

نتایج روی داده تست:

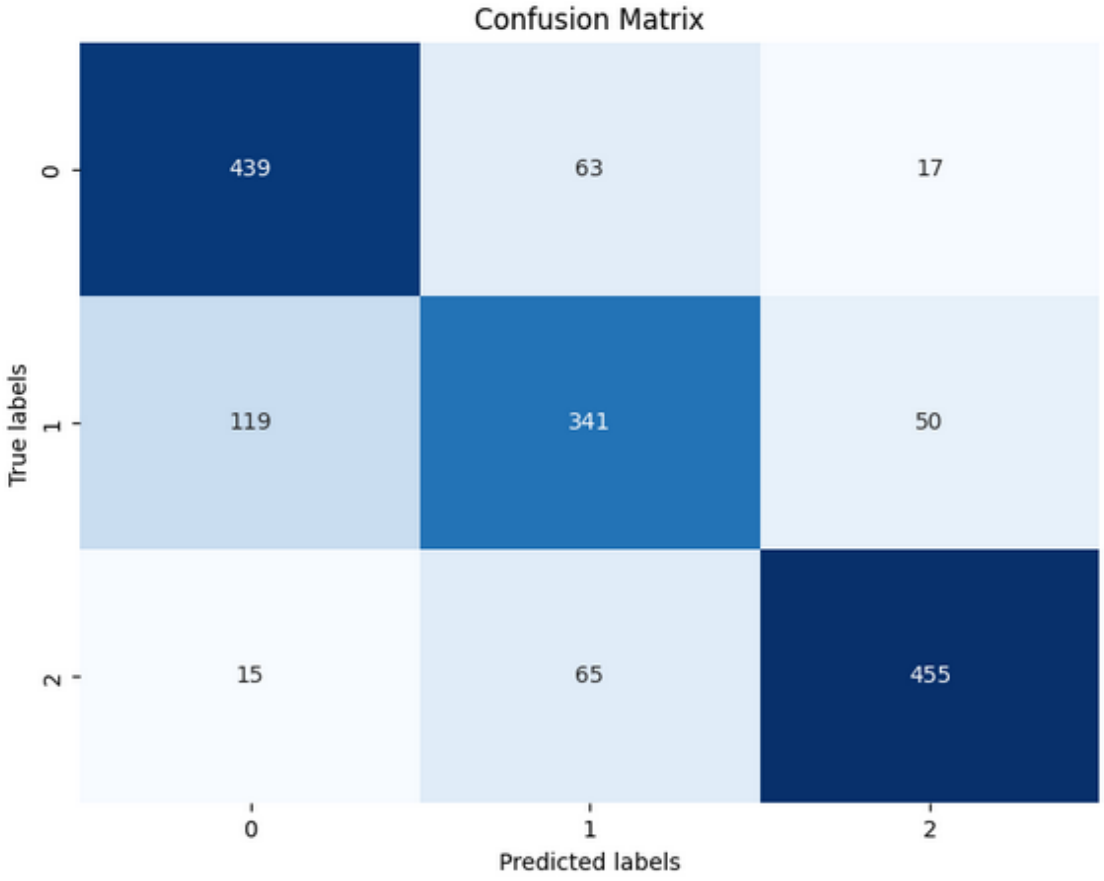
```
trainer.evaluate(test_dataset)
```

```
[98/
/usr/local/lib/python3.10/dist-packages/
You can avoid this message in future by
Passing `trust_remote_code=True` will be
  warnings.warn(
{'eval_loss': 0.5527648329734802,
 'eval_accuracy': 0.7896419437340153,
```

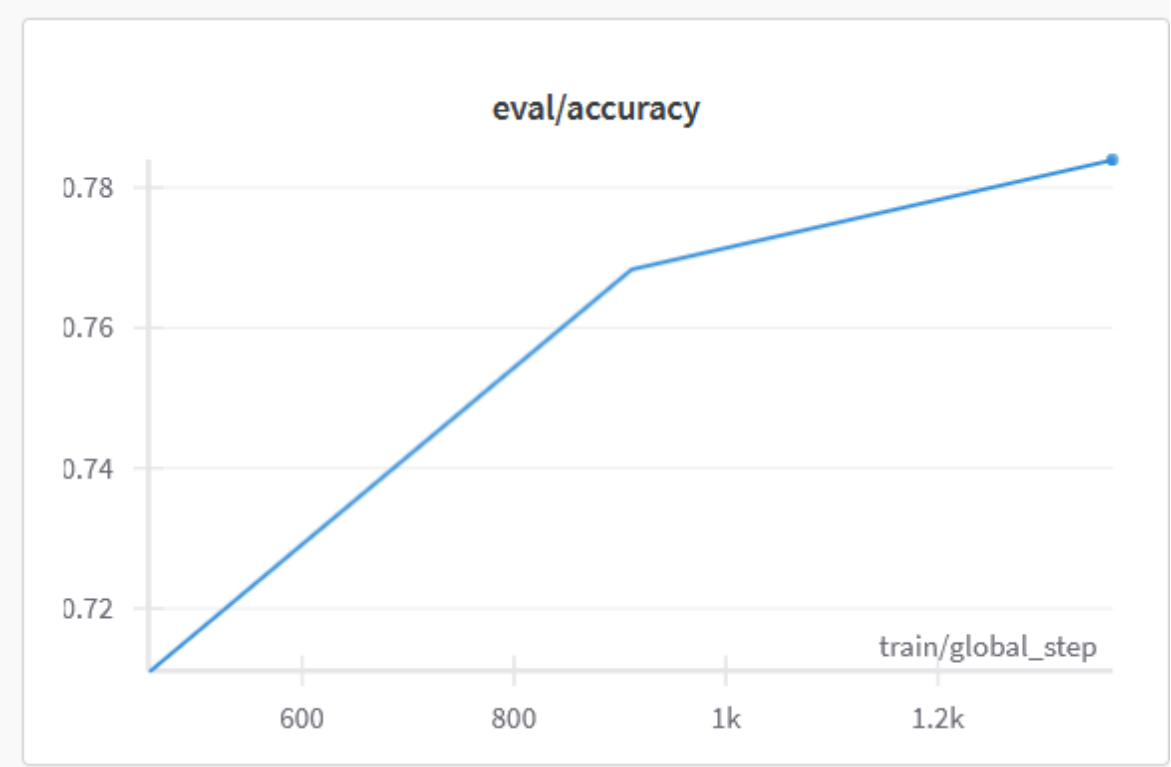
معیار های خواسته شده :

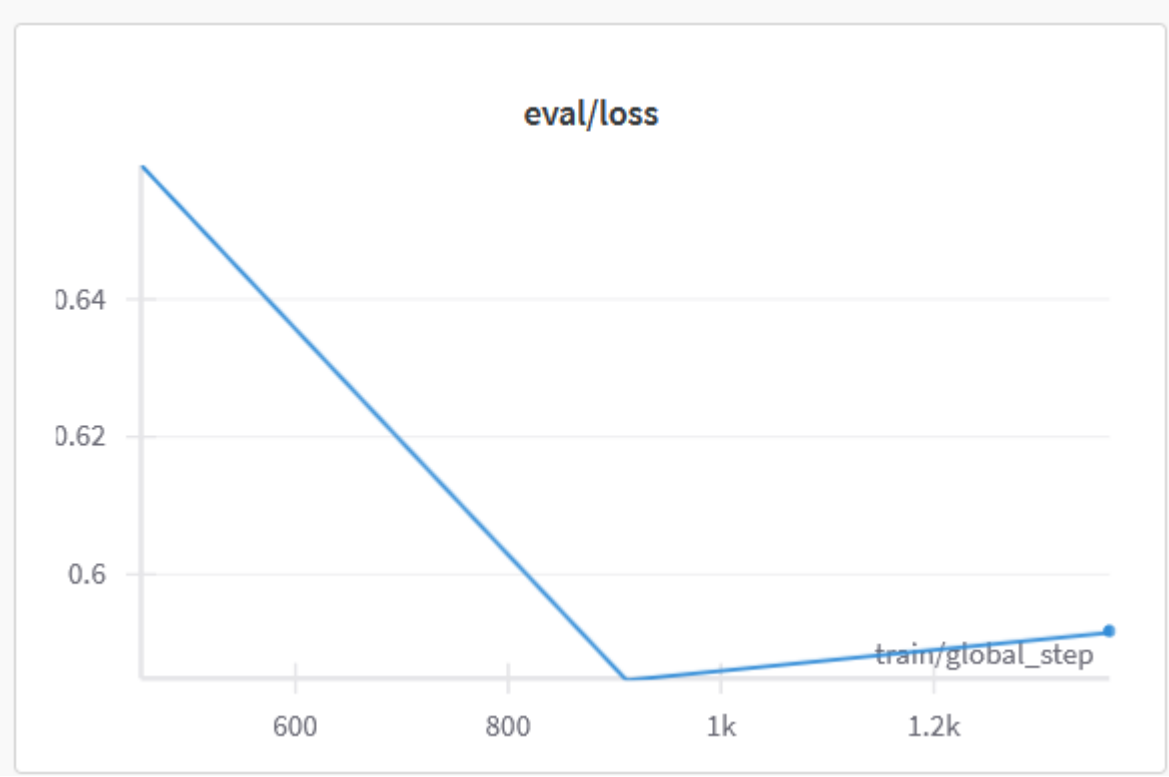
	precision	recall	f1-score	support
0	0.77	0.85	0.80	519
1	0.73	0.67	0.70	510
2	0.87	0.85	0.86	535
accuracy			0.79	1564
macro avg	0.79	0.79	0.79	1564
weighted avg	0.79	0.79	0.79	1564

Confusion matrix



نمودار های دقت و خطا روی داده ارزشیابی :


eval/accuracy
0.78
0.76
0.74
0.72
train/global_step
600
800
1k
1.2k


eval/loss
0.64
0.62
0.6
train/global_step
600
800
1k
1.2k

فریز کردن تمام ۱۲ لایه :

همه ۱۲ لایه را مطابق زیر فریز می کنیم و مجدد مشابه قبل نمودار ، معیار ها و ... خواسته شده را رسم می کنیم

```
modules = [*model.bert.encoder.layer[:12]]
for module in modules:
    for param in module.parameters():
        param.requires_grad = False
```

نتایج :

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	0.825214	0.593364
2	0.919200	0.787108	0.621991
3	0.690600	0.817040	0.612882

نتایج روی داده تست:

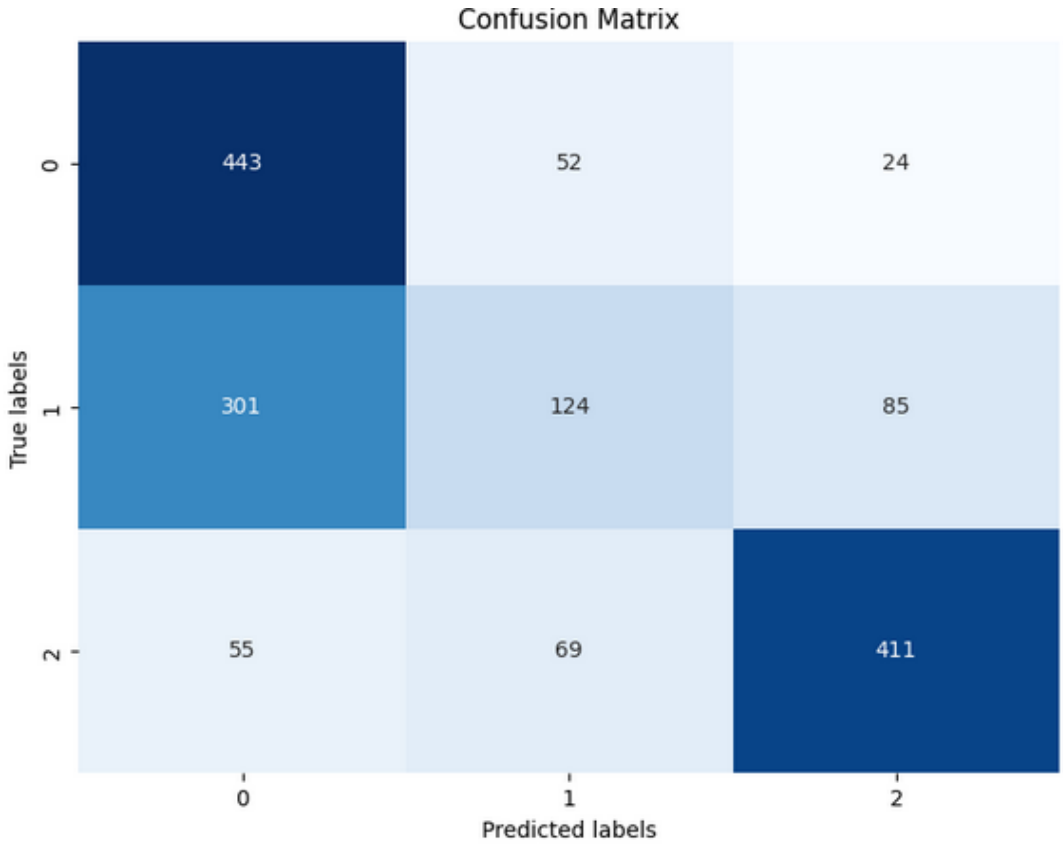
```
trainer.evaluate(test_dataset)
```

```
[98/98
/usr/local/lib/python3.10/dist-packages/d
You can avoid this message in future by p
Passing `trust_remote_code=True` will be
  warnings.warn(
{'eval_loss': 0.7709319591522217,
 'eval_accuracy': 0.6253196930946292,
```

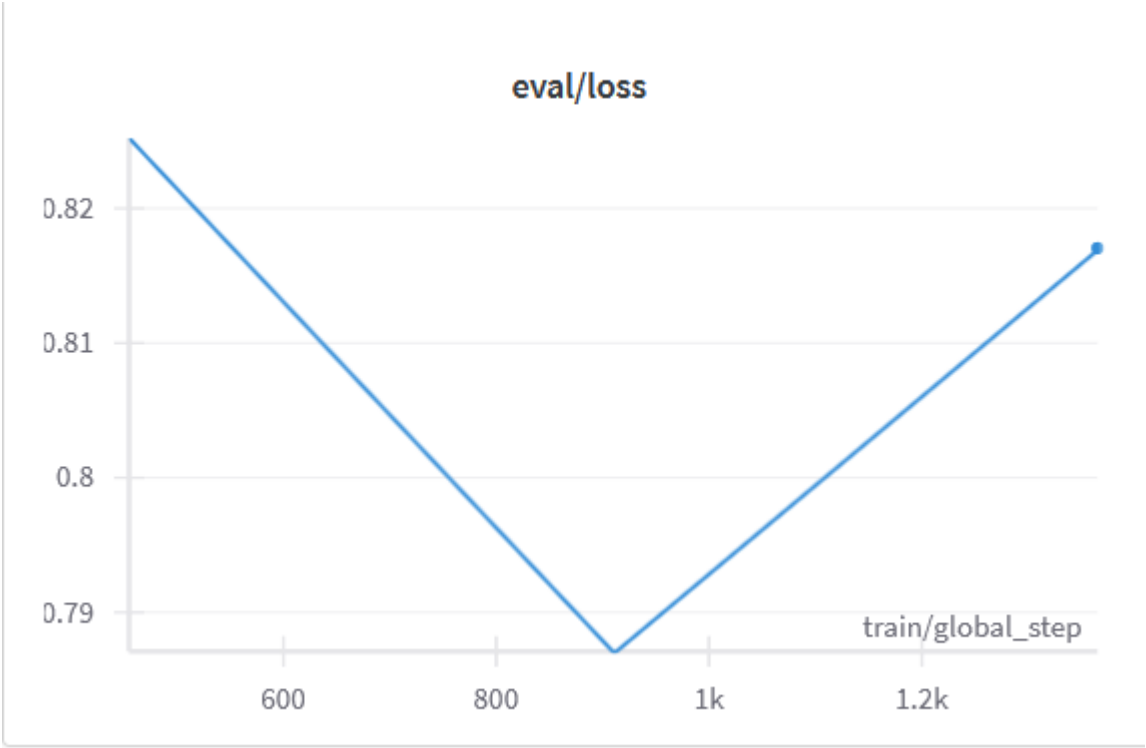
معیار های خواسته شده :

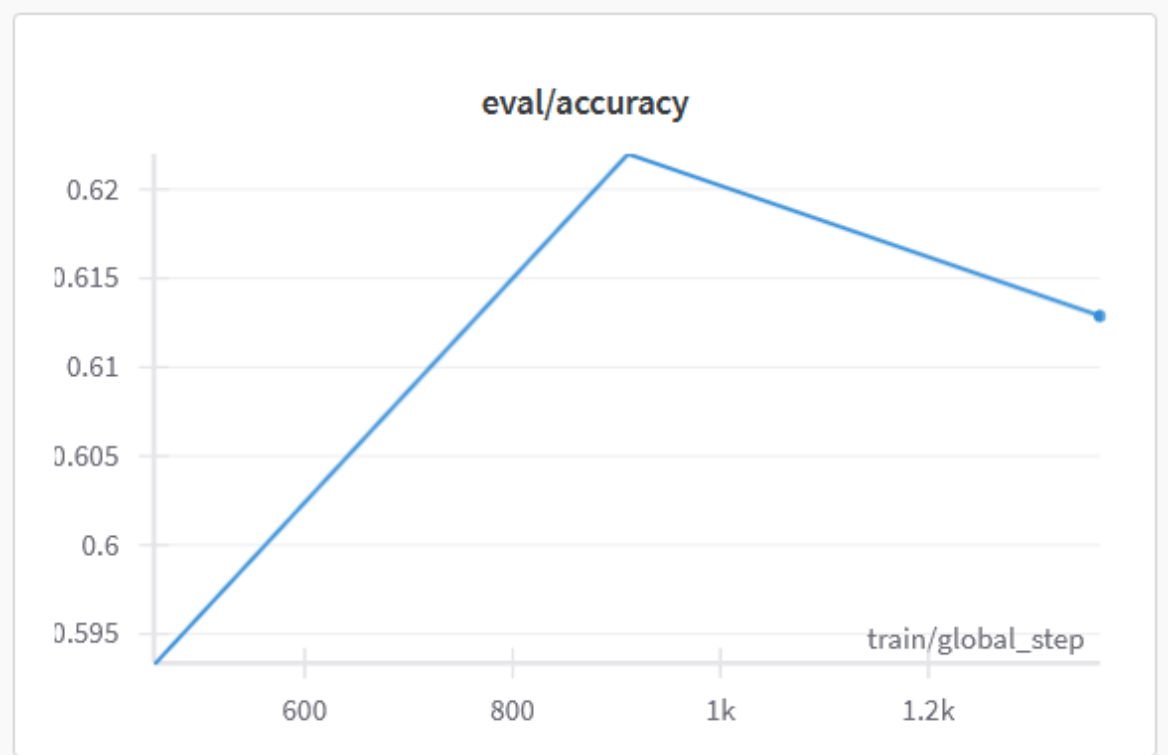
	precision	recall	f1-score	support
0	0.55	0.85	0.67	519
1	0.51	0.24	0.33	510
2	0.79	0.77	0.78	535
accuracy			0.63	1564
macro avg	0.62	0.62	0.59	1564
weighted avg	0.62	0.63	0.60	1564

Confusion matrix :



نمودار های دقت و خطا روی داده ارزشیابی :





۲-۱-۴. تنظیم دقیق مدل بر روی لایه‌های میانی

ابتدا ۹ لایه اول را نگه می داریم و بقیه را مطابق زیر حذف می کنیم تا مدل به شکلی که در ادامه گزارش شده درآید:

```
old_modules = model_copy.bert.encoder.layer
new_modules = nn.ModuleList()
for i in range(9):
    new_modules.append(old_modules[i])

model2 = copy.deepcopy(model_copy)
model2.bert.encoder.layer = new_modules

model2
```

```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(100000, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-8): 9 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=3, bias=True)
)
```

نتایج :

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	0.550534	0.770982
2	0.697500	0.609692	0.798308
3	0.338500	0.725812	0.806766

نتایج روی داده تست:

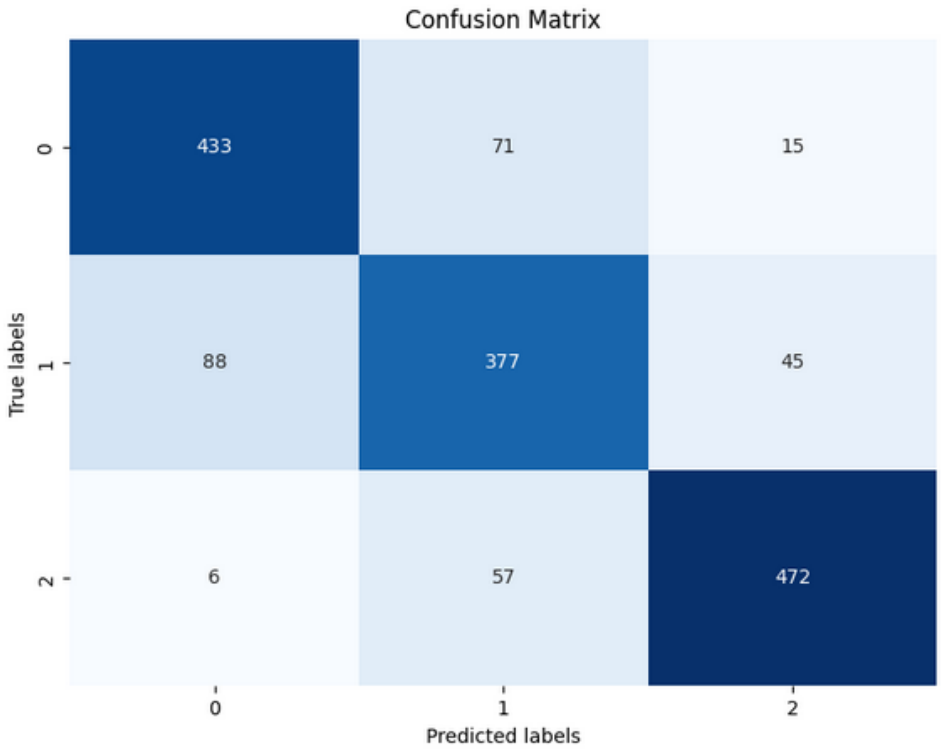
```
trainer.evaluate(test_dataset)
```

```
[98/98 00:1
/usr/local/lib/python3.10/dist-packages/datas
You can avoid this message in future by passi
Passing `trust_remote_code=True` will be manc
  warnings.warn(
{'eval_loss': 0.6709965467453003,
 'eval_accuracy': 0.819693094629156,
```

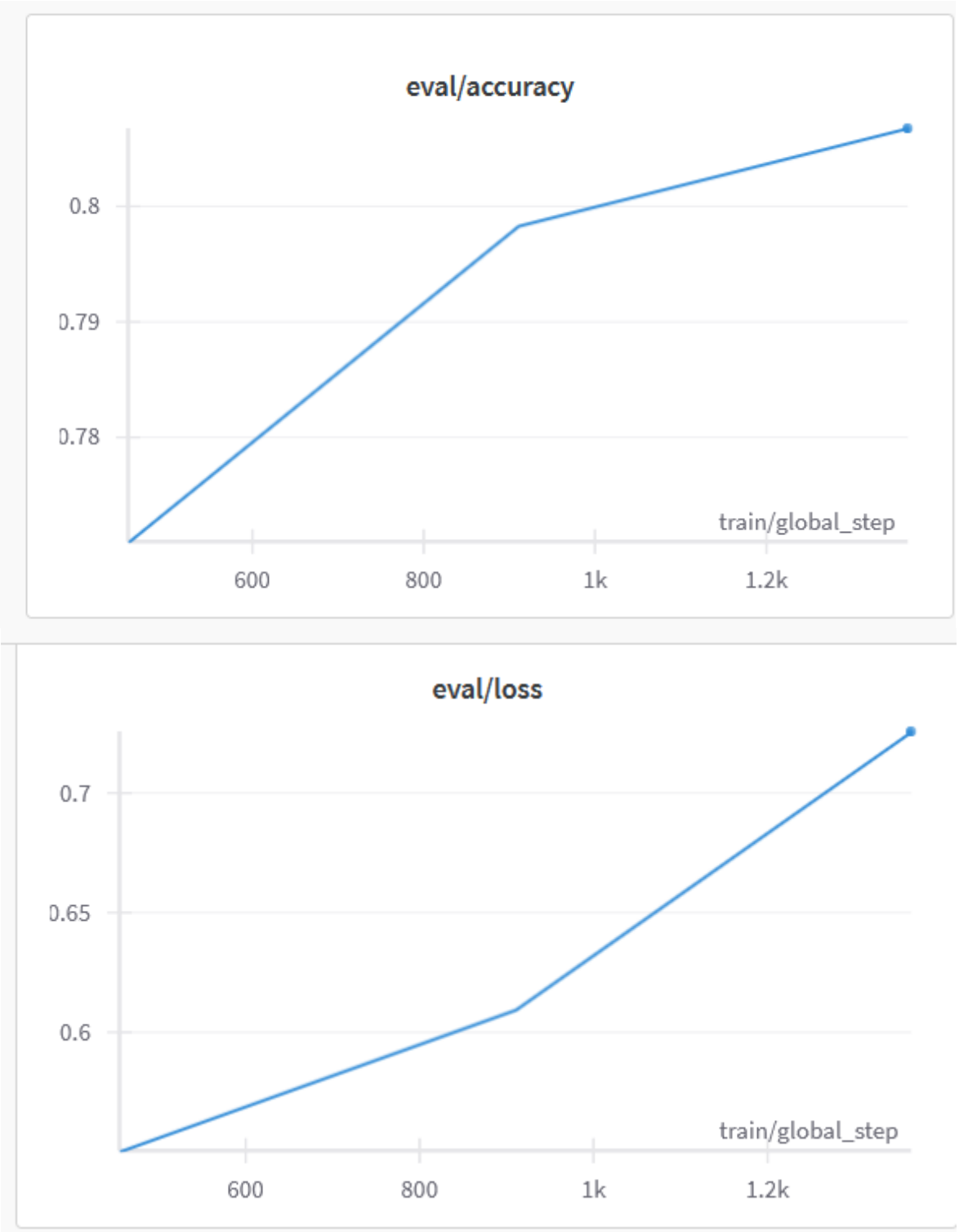
معیار های خواسته شده :

	precision	recall	f1-score	support
0	0.82	0.83	0.83	519
1	0.75	0.74	0.74	510
2	0.89	0.88	0.88	535
accuracy			0.82	1564
macro avg	0.82	0.82	0.82	1564
weighted avg	0.82	0.82	0.82	1564

Confusion matrix :



نمودار های دقت و خطا روی داده ارزشیابی :



تحلیل نتایج در انتها به طور کامل بیان شده است.

۲-۱-۵. حذف headهای attention در مدل

پارامتر های مدل :

```
Parameter name: bert.embeddings.word_embeddings.weight
Parameter name: bert.embeddings.position_embeddings.weight
Parameter name: bert.embeddings.token_type_embeddings.weight
Parameter name: bert.embeddings.LayerNorm.weight
Parameter name: bert.embeddings.LayerNorm.bias
Parameter name: bert.encoder.layer.0.attention.self.query.weight
Parameter name: bert.encoder.layer.0.attention.self.query.bias
Parameter name: bert.encoder.layer.0.attention.self.key.weight
Parameter name: bert.encoder.layer.0.attention.self.key.bias
Parameter name: bert.encoder.layer.0.attention.self.value.weight
Parameter name: bert.encoder.layer.0.attention.self.value.bias
Parameter name: bert.encoder.layer.0.attention.output.dense.weight
Parameter name: bert.encoder.layer.0.attention.output.dense.bias
Parameter name: bert.encoder.layer.0.attention.output.LayerNorm.weight
Parameter name: bert.encoder.layer.0.attention.output.LayerNorm.bias
Parameter name: bert.encoder.layer.0.intermediate.dense.weight
Parameter name: bert.encoder.layer.0.intermediate.dense.bias
Parameter name: bert.encoder.layer.0.output.dense.weight
Parameter name: bert.encoder.layer.0.output.dense.bias
Parameter name: bert.encoder.layer.0.output.LayerNorm.weight
Parameter name: bert.encoder.layer.0.output.LayerNorm.bias
Parameter name: bert.encoder.layer.1.attention.self.query.weight
Parameter name: bert.encoder.layer.1.attention.self.query.bias
Parameter name: bert.encoder.layer.1.attention.self.key.weight
Parameter name: bert.encoder.layer.1.attention.self.key.bias
Parameter name: bert.encoder.layer.1.attention.self.value.weight
Parameter name: bert.encoder.layer.1.attention.self.value.bias
Parameter name: bert.encoder.layer.1.attention.output.dense.weight
Parameter name: bert.encoder.layer.1.attention.output.dense.bias
Parameter name: bert.encoder.layer.1.attention.output.LayerNorm.weight
Parameter name: bert.encoder.layer.1.attention.output.LayerNorm.bias
Parameter name: bert.encoder.layer.1.intermediate.dense.weight
Parameter name: bert.encoder.layer.1.intermediate.dense.bias
Parameter name: bert.encoder.layer.1.output.dense.weight
Parameter name: bert.encoder.layer.1.output.dense.bias
Parameter name: bert.encoder.layer.1.output.LayerNorm.weight
Parameter name: bert.encoder.layer.1.output.LayerNorm.bias
Parameter name: bert.encoder.layer.2.attention.self.query.weight
Parameter name: bert.encoder.layer.2.attention.self.query.bias
Parameter name: bert.encoder.layer.2.attention.self.key.weight
Parameter name: bert.encoder.layer.2.attention.self.key.bias
Parameter name: bert.encoder.layer.2.attention.self.value.weight
Parameter name: bert.encoder.layer.2.attention.self.value.bias
Parameter name: bert.encoder.layer.2.attention.output.dense.weight
Parameter name: bert.encoder.layer.2.attention.output.dense.bias
Parameter name: bert.encoder.layer.2.attention.output.LayerNorm.weight
Parameter name: bert.encoder.layer.2.attention.output.LayerNorm.bias
Parameter name: bert.encoder.layer.2.intermediate.dense.weight
Parameter name: bert.encoder.layer.2.intermediate.dense.bias
Parameter name: bert.encoder.layer.2.output.dense.weight
Parameter name: bert.encoder.layer.2.output.dense.bias
Parameter name: bert.encoder.layer.2.output.LayerNorm.weight
Parameter name: bert.encoder.layer.2.output.LayerNorm.bias
Parameter name: bert.encoder.layer.3.attention.self.query.weight
Parameter name: bert.encoder.layer.3.attention.self.query.bias
Parameter name: bert.encoder.layer.3.attention.self.key.weight
Parameter name: bert.encoder.layer.3.attention.self.key.bias
Parameter name: bert.encoder.layer.3.attention.self.value.weight
```

- برای حذف **attention head** از **prune** از **torch.nn.utils** استفاده می کنیم و پارامتر هایی که شامل **attention** در نام خود هستند و با **weight** تمام می شوند را به طور رندوم ۵۰ درصد آن ها را حذف می کنیم.

نتایج :

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	0.650630	0.782043
2	0.355800	0.949961	0.765777
3	0.181100	1.147190	0.780091

نتایج روی داده تست:

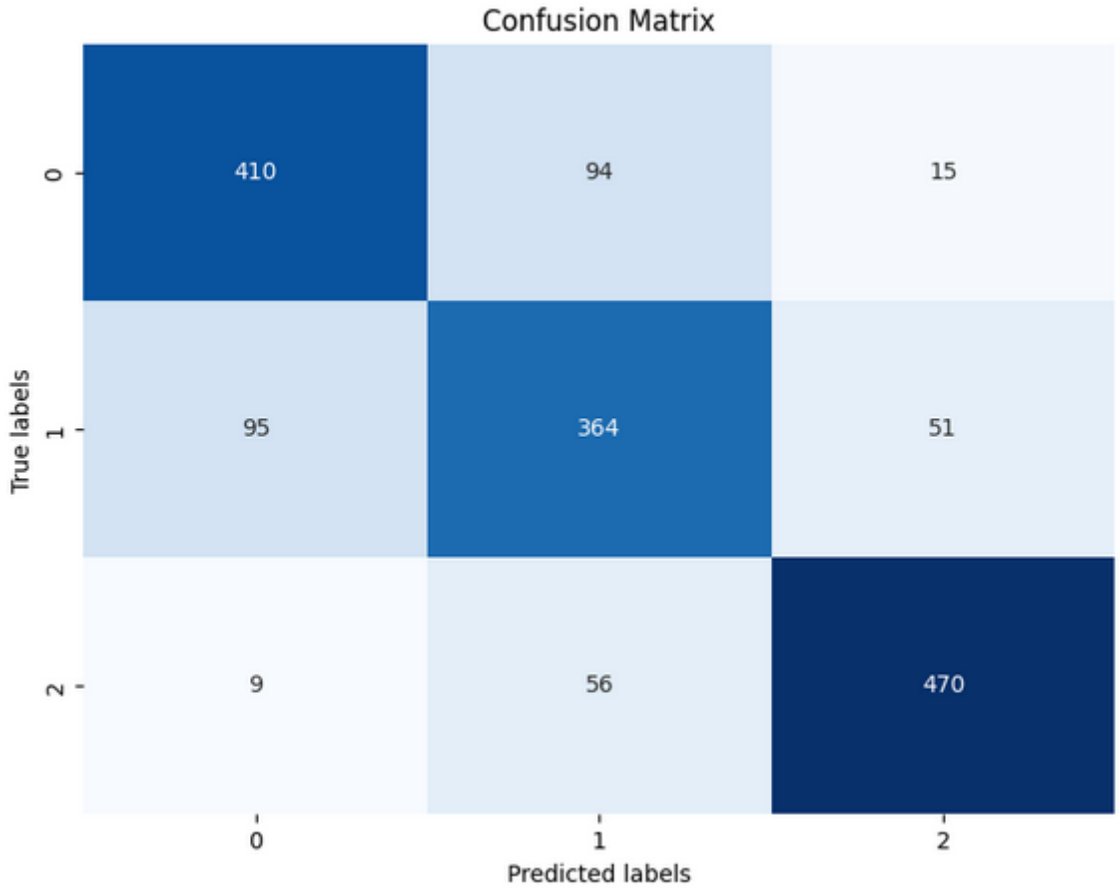
```
[84] trainer.evaluate(test_dataset)
```

```
[98/98
/usr/local/lib/python3.10/dist-packages/da
You can avoid this message in future by pa
Passing `trust_remote_code=True` will be n
  warnings.warn(
{'eval_loss': 0.5734803676605225,
 'eval_accuracy': 0.7953964194373402,
```

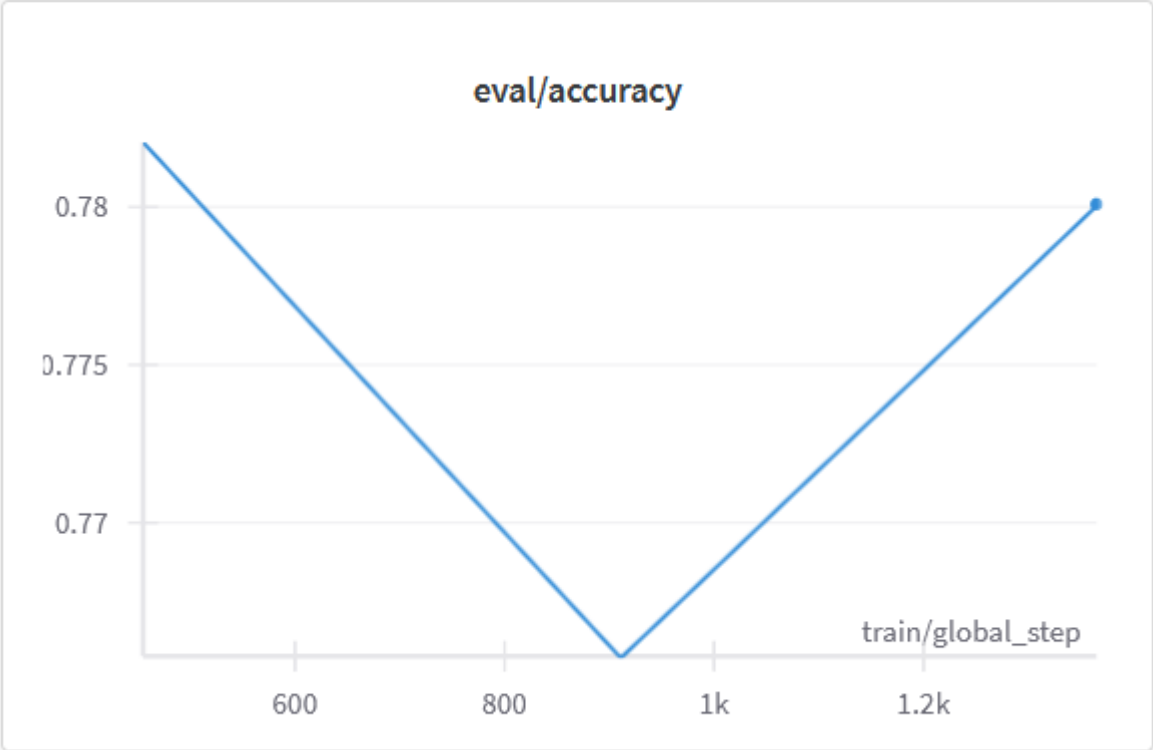
معیار های خواسته شده :

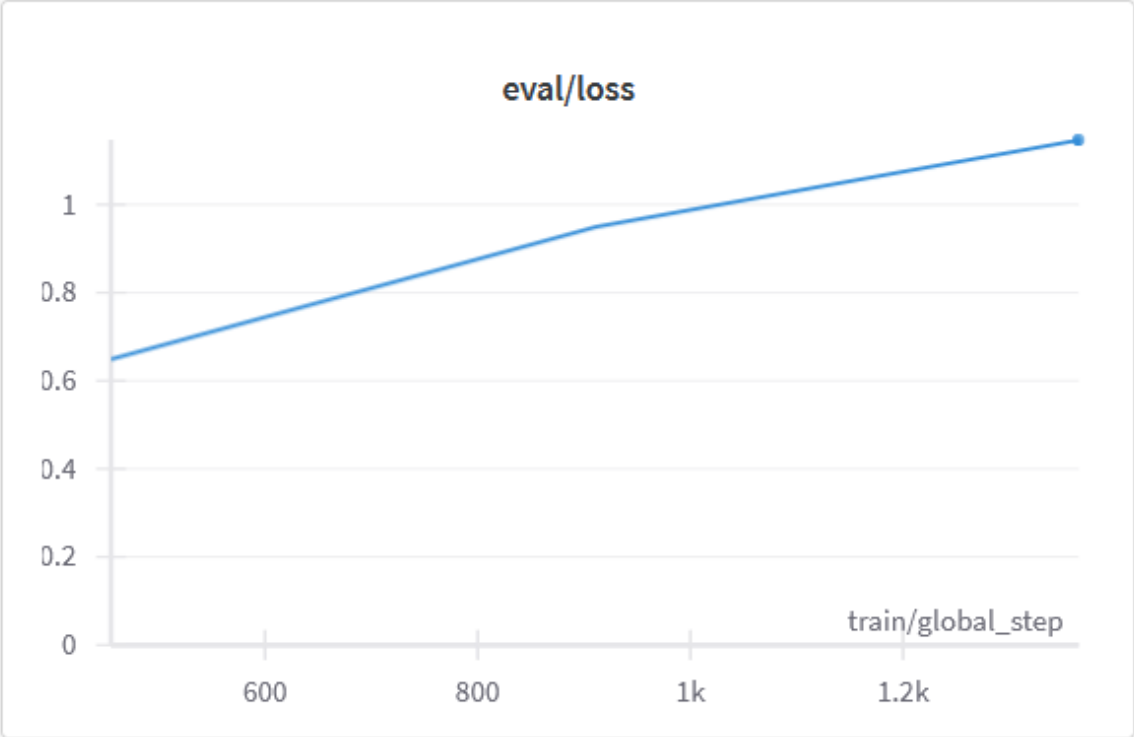
	precision	recall	f1-score	support
0	0.80	0.79	0.79	519
1	0.71	0.71	0.71	510
2	0.88	0.88	0.88	535
accuracy			0.80	1564
macro avg	0.79	0.79	0.79	1564
weighted avg	0.80	0.80	0.80	1564

Confusion matrix :



نمودار های دقت و خطا روی داده ارزشیابی :


eval/accuracy
0.78
0.775
0.77
train/global_step
600
800
1k
1.2k


eval/loss
1
0.8
0.6
0.4
0.2
0
train/global_step
600
800
1k
1.2k

تحلیل نتایج :

	Val acc	Test acc	Train loss	Val loss	Test loss	F1 score
Fine tune all layers	**0.81**	**0.83**	**0.32**	**0.74**	**0.69**	**0.82**
Freeze 9 layers	**0.78**	**0.78**	**0.53**	**0.59**	**0.55**	**0.79**
Freeze all 12 layers	**0.61**	**0.63**	**0.69**	**0.82**	**0.77**	**0.59**
Delete middle layers	**0.81**	**0.82**	**0.34**	**0.73**	**0.67**	**0.82**
Prune attention heads	**0.78**	**0.8**	**0.18**	**1.14**	**0.57**	**0.79**

مطابق جدول فوق می بینیم با در نظر گرفتن تمامی معیار ها مدلی که برخی از لایه های میانی آن را کاهش دادیم عملکرد بهتری در مقایسه با دیگر مدل ها داشته است چرا که به نظر می آید مدل bert مدل سنگینی بوده است و با سبک تر کردن آن می توان عملکرد بهتر آن را روی مخصوصا داده تست مشاهده کرد البته باید به نقش attention head ها توجه داشت چرا که سبک کردن مدل با حذف آن ها به خوبی زمانی که لایه های مدل را کم کردیم عمل نکرده است و این گویا تاثیر بسیار زیاد مدل های ترسفورمری و عملکرد بسیار بالا آن ها است که از مدل attention بهره می برند.

پس از مدلی که برخی لایه های میانی آن را حذف کردیم مدلی که تمام پارامتر های آن را از اول ترین کردیم بدون فریز عملکرد بهتری دارد ، چرایی این نتیجه واضح است چرا که کل مدل از اول آموزش دیده و با دیتا ست ما مچ شده است منتها باید به فاصله این دقت بدست آمده با زمانی که تعداد ۹ لایه را فریز می کنیم توجه کرد ، مشاهده می کنیم مدلی که ۹ لایه آن را فریز کردیم هم عملکرد نسبتا خوبی با اختلاف ۱ تا ۲ درصد در هر کدام از خانه ها جدول عمل کرده است در حالی که مجبور نبودیم همه لایه های آن را از اول آموزش بدهیم و زمان کوتاه تری برای آموزش صرف کردیم.

همچنین مشاهده کردیم که فریز کردن تمامی ۱۲ لایه اصلا عملکرد مناسبی ندارد چرا که مدل فرصت نمی کند متناسب با دیتاست ما خود را تنظیم کند و باید همواره تعداد لایه کافی درجه آزادی برای مدل قرار داد تا با دیتاست مورد نظر تنظیم شود.

در هنگام ترین کردن نمودارهای loss روی validation set گاها صعودی بودند که به نظر می آید ۳ ایپاک برای اینکه validation به حالت ایستا دقت و خطا برسد مناسب نیست ولی مطابق مقاله ۳ قرار دادیم.