گزارش پروژه تحلیل احساسات – محمد اصولیان ۹۹۵۲۱۰۷۳- محمدحسین عباس‌پور ۹۹۵۲۱۴۳۳

مقدمه

در این پروژه وظیفه ما نوشتن یک مدل هوش مصنوعی پردازش متن برای تحلیل احساسات بود. در واقع به ما یک سری جمله کوتاه داده میشود و ما باید حالت احساسی نویسنده را از روی متن جمله تشخیص دهیم. این حالات احساسی شامل ناراحت، خوشحال، عصبانی، متعجب، متنفر، ترسیده و غیره بودند.

مجموعه دادگان

برای آموزش مدل نیاز به داده‌های نمونه ای داشتیم که مدل، الگو را از این داده های نمونه یاد بگیرد. چنین دیتاستی از قبل توسط افراد دیگر آماده شده بوده و در این آدرس در دسترس است. این داده‌ها داده های ArmanEmo نام دارند که شامل یک سری جملات هستند که از متون فارسی و توییتر فارسی جمع آوری شده اند. در هر نمونه این مجموعه داده، یک جمله و یک احساس از هفت احساس نام برده شده در بالا وجود دارد که وضعیت جمله را نشان میدهد.

این مجموعه داده شامل یک دیتاست برای train و یک دیتاست test است. مجموعه آموزش، ۶۱۵۷ و مجموعه تست، ۱۱۴۹ داده دارد.

تعداد از نمونه‌های مجموعه train برای درک بهتر قرار داده شده:

```
SAD   خیلی کوچیک هستن و سایزشون بدرد نمیخوره میخوام پس بدم
HATE  از صدای پرنده دم دمای صبح متنفرم متنفرم متنفرم
```

پیش پردازش داده ها

برای پیش پردازش داده ها، لازم بود که نمونه ها را به شکلی در بیاوریم که برای مدل قابل پردازش باشد. حالت احساسات نوشته شده برای هر نمونه برای مدل قابل پردازش نیست به همین علت لازم بود که حالات احساسات را به فرمت one_hot در بیاوریم.

```
classes = ['HAPPY', 'SAD', 'ANGRY', 'FEAR', 'SURPRISE', 'HATE', 'OTHER']
#           0       1      2        3       4           5       6
```

برای این کار ابتدا به هر حالت احساسی یک id دادیم و سپس ایدی ها را به شکل آرایه ای ۷ تایی در آوردیم که خانه مربوط به id در آن ۱ است و بقیه خانه ها صفر هستند. مثلا ارایه آیدی های ۲ و ۰ به این صورت هستند:

```
[.0 .0 .1 .0 .0 .0 .0]
[.1 .0 .0 .0 .0 .0 .0]
```

همچنین برای سادگی کار با نمونه ها، به هر نمونه یک id اختصاص دادیم تا برای error checking بتوانیم راحت تر با دیتاست کار کنیم.
برای پیش پردازش داده های یک نوت بوک جدا نوشته ایم. در نهایت هم داده ها را با فرمت tsv ذخیره کردیم که به راحتی قابل لود کردن و

پردازش است. علت انتخاب فرمت tsv به حای csv این بود که در اکثر متن ها از کاراکتر (,) استفاده میشود اما از (t\) استفاده نمیشود. داده ها پس از پیش پردازش به این شکل شدند:

ID	Text	Label
0	[.0 .0 .0 .0 .0 .1 .0]	خیلی کوچیک هستن و سایزشون بدرد نمیخوره میخوام پس بدم
1	[.0 .1 .0 .0 .0 .0 .0]	از صدای پرنده دم دمای صبح متنفرم متنفرم متنفرم

علت جا به جا بودن ستون نوشته و لیبل، چپ چین بودن ادیتور است و گرنه هنگام نوشتن و خواندن ترتیب حفظ میشود و مشکلی ندارد.

مدل‌های موجود

برای حل این مسئله باید توجه داشته باشیم که زبان جملات فارسی هستند بنابراین تنها از مدل هایی میتوانستیم استفاده کنیم که چندزبانه هستند یا روی زبان فارسی train شده اند. مدل پایه اکثر این مدل ها bert و Roberta بودند. برخی از مدل ها را توضیح میدهم:

XLM with language embeddings

این دسته از مدل ها مدل هایی هستند که برای هر زبان یک embedding جدا دارند.بنابراین تنها مراحل تبدیل متن به embedding vector آنها با هم فرق میکند. مشکل این مدل های این بود که زبان‌های کمی را پوشش میدادند. یعنی زبان های کمی داشتند که برای آنها یک embedding مخصوص داشته باشند. برای مثال زبان های زیر شامل میشدند که فارسی در آنها نیست:

- xlm-mlm-ende-1024 (Masked language modeling, English-German)
- xlm-mlm-enfr-1024 (Masked language modeling, English-French)
- xlm-mlm-enro-1024 (Masked language modeling, English-Romanian)
- xlm-mlm-xnli15-1024 (Masked language modeling, XNLI languages)
- xlm-mlm-tlm-xnli15-1024 (Masked language modeling + translation, XNLI languages)
- xlm-clm-enfr-1024 (Causal language modeling, English-French)
- xlm-clm-ende-1024 (Causal language modeling, English-German)

BERT multilingual base model

این مدل، در واقع مدل bert است که روی ۱۰۰ زبان مختلف دنیا به صورت self supervised و بدون دخالت و لیبل زنی انسان آموزش دیده. نحوه آموزش هم از طریق Masked Language Modeling بوده و داده های آموزشی این مدل، صفحات ویکیپدیا بوده اند. این مدل هم زبان فارسی را پوشش میداد.

XLM-roberta

این مدل هم یک مدل cross language model است که از ۱۰۰ زبان مختلف پشتیبانی میکند. Xlm-rebrta هم مثل مدل قبلی از طریق روش MLM که یک روش self supervised است، روی ۲.۵ ترابایت داده متون موجود در اینترنت آموزش دیده است.

bert-base-parsbert-uncased

این مدل یک مدل تک زبانه است که در سال ۲۰۰۵ برای زبان فارسی توسعه یافته. پایه اصلی این مدل، مدل BERT-base است که روی داده های فارسی train شده. این گزینه هم به علت توسعه یافته بودن برای داده های فارسی گزینه خوبی برا تسک ما محسوب میشد.

مدل انتخاب شده

برای انجام این تسک، از بین مدل های موجود روی دو مدل مختلف کار کردیم. ابتدا روی مدل bert-base-parsbert کار کردیم و پس آن سراغ مدل xlm-reberta رفتیم.

Bert base parsbert

برای کار با این مدل ابتدا داده هایی که داشتیم را بدون تغییر خاصی در مدل یا داده ها(به غیر از preprocessing) روی مدل تست کردیم. دقت مدل به ۶۱ درصد رسید اما به شدت روی داده های Train، overfit می‌شد. در ادامه سعی کردیم با اصلاح داده ها و همچنین ایجاد تغییرات کوچک در مدل، از شدت overfit شدن مدل کم کنیم تا دقت بیشتر شود. در نهایت دقت مدل پس از انجام اصلاحات، به ۶۵ درصد رسید.

Xlm-roberta

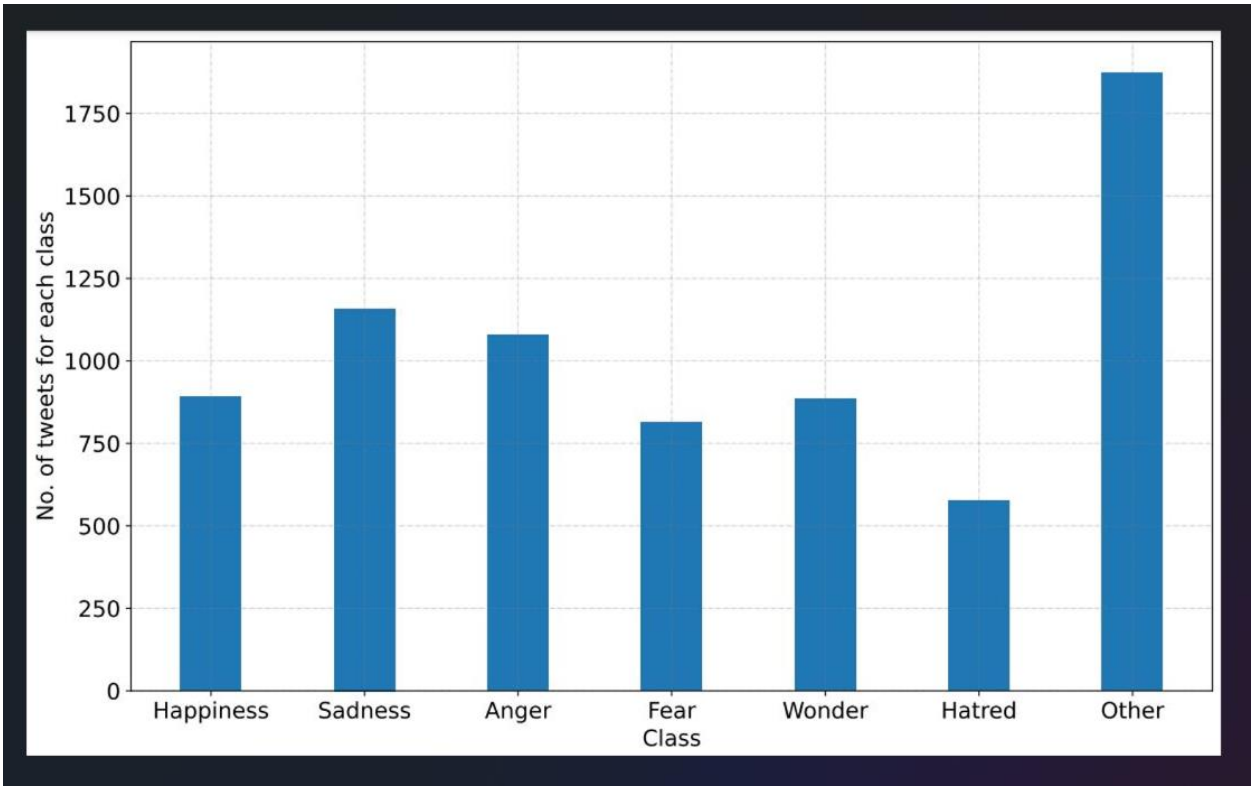
برای این مدل هم دو نسخه xlm-roberta-base وجود داشت که نسخه سبک تر بود و مدل xlm-roberta-large که نسخه سنگین تر بود. داده ها را ابتدا به صورت خام روی مدل fine tune کردیم تا نتایج را ببینیم. دقت در مدل base، به ۶۲ درصد رسید اما به علت کمبود وقت فرصت کار بیشتر روی این مدل را نداشتیم.

مقاله مجموعه دادگان

توضیح نمونه ها

مقاله ARMANEMO: A Persian Dataset for Text-Based Emotion Detection، یک مجموعه داده جدید به ما معرفی میکند که برای تسک های nlp روی زبان فارسی مناسب است.

دیتاست معرفی شده در این مقاله شامل نمونه هایی است که احساسات یک جمله را نشان میدهند. در واقع یک مجموعه داده برای تسک دسته بندی جمله است. احساسات پوشش داده شده در این مقاله شامل شادی، خشم، غم، تعجب، وحشت، تنفر و سایر است. به هر نمونه دراین دیتاست تنها یکی از این لیبل ها داده میشود. توزیع لیبل ها در دیتاست به این صورت است:



این دیتاست شامل بیشتر از ۷۰۰۰ نمونه است که درهر نمونه، یک جمله فارسی و یک لیبل برای آن وجود دارد.

مجموعه داده های این دیتاست از شبکه های اجتماعی فارسی مثل اینستاگرام، توییتر، کامنت های دیجی کالا و ... جمع آوری شده اند.

نحوه لیبل زنی

برای لیبل زدن این داده ها از افرادی استفاده شده که زبان مادری فارسی دارند و به آن مسلط هستند. برای دقت بیشتر در لیبل زنی به هر یک از افراد یک سری راهنما ها و دستورات داده شده تا طبق این دستورات لیبل زنی ها را انجام دهند.

در مواردی که لیبل هایی که افراد وارد کرده اند، با هم دیگر تداخل وجود دارد، روی تداخل ها اجماع صورت گرفته تا همگی به یک نظر واحد برای نمونه برسند و سپس لیبل تعیین شده. در صورتی که افراد به یک نظر واحد نرسیده باشند، لیبل رد شده و در دیتاست قرار نگرفته.

درنهایت هم روی داده های دیتاست عملیات پاکسازی صورت گرفته یعنی داده هایی که ایراد نگارشی یا گرامری داشتند اصلاح شده اند.

مقاله های استفاده شده

ARMANEMO: A Persian Dataset for Text-Based Emotion Detection

این مقاله همان مقاله مجموعه داده هاست که در بالاتر توضیح داده شده. مدل ما روی این مجموعه ترین و تست شده.

ParsBERT: Transformer-based Model for Persian Language Understanding

این مقاله، مدل پارس برت را معرفی میکند که در بالاتر توضیح داده شده. ما از مدل پارس برت برای رسیدن به دقت بالای ۶۵ درصد استفاده کردیم.

Unsupervised Cross-lingual Representation Learning at Scale

مدل xlm-roberta هم در این مقاله معرفی شد که در بالاتر توضیح داده شده. از این مدل هم برای آموزش داده ها استفاده کردیم اما به نتایج خوبی نرسیدیم.

شرح پیاده سازی ها و اقدامات انجام شده

در این قسمت مراحل مختلف گذرانده شده تا رسیدن به نتایج نهایی را توضیح میدهم

Preprocessing

برای پیشپردازش داده ها از یک فایل نوتبوک جدا به اسم Preprocess.ipynb استفاده کردم.

ابتدا داده ها را از گیتهاب دیتاست دانلود کردم:

```
! cd content
! git clone https://github.com/Arman-Rayan-Sharif/arman-text-emotion.git
```

سپس کلاس ها را با یک ترتیب خاص در یک لیست قرار دادم. ایندکس هر کلاس در لیست معادل آیدی همان کلاس میشود. از روی لیست ساخته شده دو دیکشنری هم ساختم که اسم کلاس را به id و id را به اسم کلاس تبدیل میکند:

```
[ ]  classes = ['HAPPY', 'SAD', 'ANGRY', 'FEAR', 'SURPRISE', 'HATE', 'OTHER']
     class2id = {classes[i]: i for i in range(len(classes))}
     id2class = {i: classes[i] for i in range(len(classes))}
```

همچنین تابعی نوشتم که آیدی را به فرمت one_hot تبدیل میکند:

```
def class2onehot(classname):
  classid = class2id[classname]
  result = np.zeros(len(classes))
  result[classid] = 1
  return result
```

سپس تابع اصلی را نوشتم که آدرس فایل ورودی و خروجی را میگیرد، دیتاست را از ادرس ورودی میخواند، لیبل ها را به one_hot تبدیل میکند و به هر نمونه یک id هم اختصاص میدهد و در نهایت داده های تغییر یافته را با فرمت tsv در خروجی مینویسد:

```
def preprocess_file(inpath, outpath):
  with open(inpath, 'r') as f:
    lines = f.readlines()
  texts = []
  labels = []
  ids = []

  for i, line in enumerate(lines):
    line = line.strip()
    text, classname = line.split('\t')
    id = i
    texts.append(text)
    labels.append(class2onehot(classname))
    ids.append(id)

  with open(outpath, 'w') as f:
    f.write('ID\tText\tLabel\n')
    for i in range(len(ids)):
      f.write(f'{ids[i]}\t{texts[i]}\t{labels[i]}\n')
```

سپس این تابع را روی فایل داده های تست و ترین اجرا کردم:

```
preprocess_file('arman-text-emotion/dataset/train.tsv', 'pptrain.tsv')
preprocess_file('arman-text-emotion/dataset/test.tsv', 'pptest.tsv')
```

Dadmatools:

این یک ابزاری برای پردازش متون فارسی میباشد که توسط دادماتک و بعضی از دانشجویان دانشگاه علم و صنعت درست شده است. در این پروژه فقط از Normalizer این ابزار استفاده شده است؛ به این صورت که فاصله های اضافی و آدرس سایت و ایموجی و ... را با این ابزار حذف میکنیم:

```
normalizer = Normalizer(
    full_cleaning=False,
    unify_chars=True,
    refine_punc_spacing=True,
    remove_extra_space=True,
    remove_puncs=False,
    remove_html=True,
    remove_stop_word=False,
    replace_email_with="<EMAIL>",
    replace_number_with=None,
    replace_url_with="",
    replace_mobile_number_with=None,
    replace_emoji_with="",
    replace_home_number_with=None
)
```

از این بخش قبل از tokenize کردن متن استفاده کردیم.

آموزش مدل

برای آموزش مدل ها ابتدا مدل xlm-roberta-base را توضیح میدهم و سپس سراغ parsbert که جزعیات پیاده‌سازی بیشتری دارد میروم:

پیاده سازی روبرتا

برای پیاده سازی این مدل از کتابخانه هاگینگ فیس استفاده کردم. وزن های ترین شده این مدل در hugging face موجود هستند. تنها لازم بود که مدل را دانلود و آنرا fine tune کنیم. همچنین برای لود کردن دیتاست و کار با دیتاست از کتابخانه datasets استفاده کردم و برای سنجش متریک ها شامل f۱ و دقت و ... هم از کتابخانه scikitlearn استفاده کردم.

در ابتدا، کتابخانه های لازم را نصب و سپس لود کردم:

```
! pip install -q accelerate datasets evaluate
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 270.9/270.9 kB 5.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 507.1/507.1 kB 10.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.1/84.1 kB 9.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 115.3/115.3 kB 10.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 10.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 10.1 MB/s eta 0:00:00
```

```
import torch
import torch.nn as nn
import numpy as np
from datasets import load_dataset, DatasetDict
import evaluate
from sklearn.metrics import f1_score, accuracy_score, recall_score, precision_score

from transformers import AutoConfig, AutoModelForSequenceClassification, AutoTokenizer
from transformers import TrainingArguments, Trainer
from transformers import DataCollatorWithPadding, EvalPrediction
from transformers.optimization import AdamW

from time import time
from transformers import set_seed
set_seed(365)
```

سپس داده های ترین و تست را لود کردم:

```
ds_url = f'/content/'
ds_files = {
    'train': ds_url + 'pptrain.tsv',
    'test': ds_url + 'pptest.tsv',
}

ds = load_dataset('csv', data_files=ds_files, delimiter='\t')
ds = ds.rename_columns({'ID': 'id', 'Text': 'text', 'Label': 'label'})
```

ستون های دیتاست لود شده به این صورت هستند:

```
DatasetDict({
    train: Dataset({
        features: ['id', 'text', 'label'],
        num_rows: 6125
    })
    test: Dataset({
        features: ['id', 'text', 'label'],
        num_rows: 1151
    })
})
```

دیتاست لود شده، مقادیر لیبل ها را به صورت استرینگ میخواند به همین علت لازم بود تابعی بنویسم که مقدار را به صورت استریک دریافت میکند و تبدیل به یک آرایه numpy میکند که برای مدل قابل پردازش است:

```
def convert_labels(example):
  # result = np.zeros((2))
  # result[example['label']] = 1
  # example['label'] = result
  example["label"] = [float(num) for num in example['label'][1:-1].split(' ')]
  return example
```

سپس این تابع را روی کل دیتاست اجرا کردم:

```
ds = ds.map(convert_labels)
ds = ds.map(replace_none_with_str)
```

در قسمت بعد باید مدل را لود کنم. برای این کار از checkpoint مدل در هاگینگ فیس استفاده کردم. برای راحتی کار، در همین قسمت هم تعداد epoch های آموزش را مشخص کردم:

```
[130] num_epochs = 5
      checkpoint = 'FacebookAI/xlm-roberta-base'
```

مدل لازم است که از توکنایزر مخصوص خود هم استفاده کند. کار توکنایزر این است که داده های متنی رابه id هایی تبدیل میکند که برای مدل قابل شناسایی است. مدل از روی این آیدی ها میتواند embedding را تولید کند.

برای همین از روی چک پوینت مشخص شده، مدل و توکنایزر آن را لود کردم و به کمک توکنایزر، دیتاست خام را توکنایز کردم:

```
] tokenizer = AutoTokenizer.from_pretrained(checkpoint)
  def tokenize_function(example):
    return tokenizer(example['text'], truncation=True, max_length=256, add_special_tokens=True)

  tokenized_datasets = ds.map(tokenize_function, batched=True)
  data_collator = DataCollatorWithPadding(tokenizer)

  model = AutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels=7)
```

سپس training argument های مدل را مشخص کردم:

```
training_args = TrainingArguments(
    run_name=f'First Run-{time()}',
    output_dir='outputs-xml', overwrite_output_dir=True,
    # auto_find_batch_size=True,
    num_train_epochs=num_epochs,
    evaluation_strategy='epoch',
    save_strategy='epoch',
    save_total_limit=5, load_best_model_at_end=True,
    push_to_hub=True,
    hub_model_id='mohammad-osoolian/DL-xlm-roberta-base10',
    hub_strategy='every_save',
    hub_private_repo=False,
    hub_token=''
)
```

آرگومنت های این تابع موارد پیچیده ای نیستند که لازم به توضیح باشد.

سپس ترینر مدل را ایجاد کردم:

```
36] trainer = Trainer(
        model,
        training_args,
        train_dataset=tokenized_datasets['train'],
        eval_dataset=tokenized_datasets['test'],
        data_collator=data_collator,
        tokenizer=tokenizer,
        compute_metrics=compute_metrics,
    )
```

در ترینر مشخص میکنیم که چه مدلی، با چه آرگومنت هایی، روی چه دیتاست هایی اجرا شود و نحوه evaluate کردن آن چطور باشد. در اینجا مشخص کردیم که evaluate مدل باید از طریق تابع compute_mterics انجام شود که در ادامه توضیح میدهم.

در این تابع ما معیارهای مختلف شامل accuracy و f۱ و precision و recall را محاسبه کردیم:

```python
def compute_metrics(eval_preds: EvalPrediction):
  logits, labels = eval_preds.predictions, eval_preds.label_ids
  predictions = onehot(np.argmax(sigmoid(logits), axis=-1))

  f1 = f1_score(labels, predictions, average=None, zero_division=0.0)
  f1 = {f'f1_C{i}': f1[i] for i in range(len(f1))}
  f1_macro = f1_score(labels, predictions, average='macro', zero_division=0.0)
  recall = recall_score(labels, predictions, average=None, zero_division=0.0)
  recall = {f'recall_C{i}': recall[i] for i in range(len(recall))}
  recall_macro = recall_score(labels, predictions, average='macro', zero_division=0.0)
  precision = precision_score(labels, predictions, average=None, zero_division=0.0)
  precision = {f'precision_C{i}': precision[i] for i in range(len(precision))}
  precision_macro = precision_score(labels, predictions, average='macro', zero_division=0.0)
  accuracy = accuracy_score(labels, predictions)
  results = {'accuracy': accuracy, 'precision_macro': precision_macro, 'recall_macro': recall_macro, 'f1
  return results
```

همانطور که گفته شد برای محاسبه از کتابخانه scikitlearn استفاده کرده ایم.

در نهایت با مشخص شدن این مقادیر زمان آموزش مدل است. مدل ۵ ایپاک ترین شد که نتایج آن به این صورت بودند:

Epoch	Training Loss	Validation Loss	Accuracy	Precision Macro	Recall Macro	F1 Macro
1	0.324700	0.326075	0.503910	0.528971	0.497542	0.486044
2	0.236600	0.324474	0.498697	0.540036	0.496795	0.483770
3	0.204100	0.282938	0.615117	0.650267	0.608334	0.607669
4	0.152900	0.297677	0.621199	0.655268	0.594882	0.603348
5	0.123100	0.309510	0.624674	0.660845	0.613727	0.620989

از نشان دادن متریک ها برای خوانایی عکس خود داری شده و گرنه محاسبه شده اند.

دقت این مدل ۶۲ درصد بود. به علت کمبود وقت نتوانستیم روی این مدل تمرکز بیشتری بگذاریم.

پیاده سازی مدل parsbert

تمام مراحل تا قبل از آموزش مدل مانند قبل میباشد و فقط checkpoint را باید تغییر داد.

این مدل هم مشابه مدل قبلی میباشد ولی با این تفاوت که تمام لایه های parsbert به جز بخش آخر یعنی pooler را freeze کردیم و سپس بعد از دادن خروجی parsbert به لایه dropout، آن را به یک لایه classifier میدهیم:

```
def forward(self, input_ids=None, attention_mask=None, labels=None):
    outputs = self.model(input_ids, attention_mask).pooler_output
    outputs = self.dropout(outputs)
    outputs = self.classifier(outputs).to(device)
    loss = None
    if labels is not None:
        loss_fct = nn.CrossEntropyLoss()
        loss = loss_fct(outputs, labels)

    return {'loss': loss, 'logits': outputs}
```

علت اینکه در تابع forward مقدار loss هم محاسبه شده است مربوط به طراحی Trainer کتابخانه transformer میباشد. چون باید خروجی آن یک دیکشنری با این کلید ها باشد.

آرگومان های آموزش هم به شکل زیر است:

```
training_args = TrainingArguments(
    run_name=f'First Run-{time()}',
    output_dir='finetuned-parsbert-uncased-ArmanEmo',
    overwrite_output_dir=True,
    auto_find_batch_size=True,
    num_train_epochs=num_epochs,
    evaluation_strategy='epoch',
    eval_steps=512,
    save_strategy='epoch',
    save_steps=512,
    save_total_limit=5,
    load_best_model_at_end=True,
    metric_for_best_model='f1_macro',
    save_safetensors=True,
    group_by_length=True,
    push_to_hub=True,
    hub_model_id='mohammadhabp/finetuned-parsbert-uncased-ArmanEmo',
    hub_strategy='all_checkpoints',
    hub_token='...',
    warmup_steps=1000,
    weight_decay=1e-2,
    seed=SEED,
    data_seed=SEED
)
```

اکثر پارامترها مشابه قبل میباشد ولی فقط weight_decay و warmup_steps به آن اضافه شده است که با tune کردن به دست آمده‌اند.

حداقل دقت

در نهایت پس از آموزش parsbert میبینیم که دقت به بالای ۶۵ درصد رسیده که حداقل دقت لازم را دارد:

Epoch	Training Loss	Validation Loss	Accuracy
1	1.566300	1.368862	0.504778
2	0.894800	1.166213	0.603823
3	0.502500	1.265785	0.660295
4	0.222000	2.232874	0.628149
5	0.089000	2.332148	0.640313

همانطور که مشخص است مدل در epoch سوم به دقت ۶۶ درصد رسیده است.

ارزیابی مدل با معیارهای مختلف

در این تابع، ما معیارهای زیر را برای مدل حساب کردیم:

- Accuracy: ساده ترین معیار که مشخص میکند چند تا از نمونه ها توسط مدل به درستی پیشبینی شده اند
- Precision: مشخص میکند که پیشبینی های مدل تا چه میزان صحیح بوده اند.
- Recall: مشخص میکند که از همه نمونه های مثبت، چه تعداد از آنها توسط مدل پیشبینی شده.
- F۱: این معیار هم برایندی از Precision و recall است.
- معیار های precision و recall و f۱ هم به صورت کلی و macro avg محاسبه شده اند، هم برای هر کلاس به صورت جدا گانه

Accuracy	Precision Macro	Recall Macro	F1 Macro	F1 C0	F1 C1	F1 C2	F1 C3	F1 C4	F1 C5	F1 C6	Recall C0	Recall C1	Reca C2
0.503910	0.528971	0.497542	0.486044	0.500000	0.478821	0.497006	0.671642	0.488688	0.214286	0.551867	0.458182	0.496183	0.536

معیار های مشخص شده برای مدل نهایی ما به این صورت بودند:

Epoch	Training Loss	Validation Loss	Accuracy	Precision Macro	Recall Macro	F1 Macro
1	1.566300	1.368862	0.504778	0.608146	0.523069	0.511415
2	0.894800	1.166213	0.603823	0.639156	0.604609	0.601078
3	0.502500	1.265785	0.660295	0.677762	0.648165	0.653855
4	0.222000	2.232874	0.628149	0.684995	0.626498	0.632037
5	0.089000	2.332148	0.640313	0.687882	0.639706	0.644780

F1 C0	F1 C1	F1 C2	F1 C3	F1 C4	F1 C5	F1 C6
0.449735	0.633257	0.365297	0.708661	0.555556	0.407080	0.460317
0.525000	0.703533	0.553333	0.710744	0.578571	0.581818	0.554545
0.727955	0.694030	0.581132	0.752137	0.617886	0.596774	0.607069
0.666667	0.714815	0.589552	0.775862	0.516129	0.612903	0.548330
0.668122	0.731884	0.608392	0.769231	0.528634	0.650000	0.557196

Recall C0	Recall C1	Recall C2	Recall C3	Recall C4	Recall C5	Recall C6
0.309091	0.530534	0.259740	0.789474	0.517241	0.353846	0.901554
0.381818	0.874046	0.538961	0.754386	0.558621	0.492308	0.632124
0.705455	0.709924	0.500000	0.771930	0.524138	0.569231	0.756477
0.567273	0.736641	0.512987	0.789474	0.386207	0.584615	0.808290
0.556364	0.770992	0.564935	0.789474	0.413793	0.600000	0.782383

Precision C0	Precision C1	Precision C2	Precision C3	Precision C4	Precision C5	Precision C6
0.825243	0.785311	0.615385	0.642857	0.600000	0.479167	0.309059
0.840000	0.588689	0.568493	0.671875	0.600000	0.711111	0.493927
0.751938	0.678832	0.693694	0.733333	0.752475	0.627119	0.506944
0.808290	0.694245	0.692982	0.762712	0.777778	0.644068	0.414894
0.836066	0.696552	0.659091	0.750000	0.731707	0.709091	0.432665

نمونه ورودی و خروجی مدل

برای نمونه دادن به مدل و مشاهده خروجی آنجایی که وزن های مدل ذخیره شده کافیست تا از pipeline در هاگینگ فیس استفاده کنیم:

```
[2] from transformers import pipeline
    classifier = pipeline("sentiment-analysis", model = "mohammad-osoolian/parsbert-finetuned")
```

```
classifier.predict('حس خیلی خوبی دارم')
```

```
[{'label': 'HAPPY', 'score': 0.9411950707435608}]
```

```
[3] classifier.predict('امروز حال من خوب نیست')
```

```
[{'label': 'SAD', 'score': 0.887709379196167}]
```

```
[4] classifier.predict('نگرانم برایش اتفاقی افتاده باشد')
```

```
[{'label': 'FEAR', 'score': 0.8856727480888367}]
```

می‌بینیم که پیشبینی ها به درستی انجام میشود.

آماده کردن توابع predict و evaluate

تابع predict:

برای پیاده سازی این تابع از همان پایپلاین در قسمت قبل استفاده کردیم.

در این تابع جمله ها از آدرس فایل ورودی خوانده میشوند و به کلاسیفایر داده میشوند. پیشبینی کلاسیفایر ریترن میشود و در صورتی که مسیر فایل خروجی هم تعیین شده باشد، خروجی ها در فایل هم نوشته میشوند.

```
[ ]  def predict(inpath, classifier, outpath=None):
       sentences = []
       with open(inpath, 'r', encoding="utf-8-sig") as f:
         for line in f.readlines():
           sentences.append(line.strip())
       predictions = classifier.predict(sentences)
       if outpath != None:
         with open(outpath, 'w') as f:
           f.write('Label, Score\n')
           for i in range(len(sentences)):
             f.write(f"{predictions[i]['label']}, {predictions[i]['score']}\n")

       return predictions
```

نمونه اجرای این تابع به این صورت است:

جمله های درون فایل ورودی:

predict_test.csv evaluate_test.csv

predict_test.csv

```
1  سلام من خوشحال هستم
2  سلام من عصبانی هستم
3  سلام من ناراحت هستم
```

خروجی تابع و مقادیر نوشته شده درفایل:

```
predict('predict_test.csv', classifier, outpath='results.csv')
```

```
[{'label': 'HAPPY', 'score': 0.9465934634208679},
 {'label': 'OTHER', 'score': 0.9452375769615173},
 {'label': 'SAD', 'score': 0.8910128474235535}]
```

results.csv

1 to 3 of 3 entries Filter

Label	Score
HAPPY	0.9465934634208679
OTHER	0.9452375769615173
SAD	0.8910128474235535

تابع evaluate:

برای تابع evaluate هم مثل تابع قبل مقادیر را از یک فایل csv یا tsv میخوانیم و جمله ها و لیبل ها را جدا میکنیم. سپس جمله ها را به کلاسیفایر میدهیم تا پیشبینی را انجام دهد. با پاس دادن پیشبینی و لیبل ها به تابع compute_metrics که از قبل برای مدل نوشته بودیم، متریک ها محاسبه و چاپ میشوند.

```python
def evaluate(inpath, delim, classifier):
  sentences = []
  labels = []
  with open(inpath, 'r', encoding="utf-8-sig") as f:
    for line in f.readlines():
      sentence, label = line.strip().split(delim)
      labels.append(class2id[label])
      sentences.append(sentence)

  predictions = classifier.predict(sentences)
  preds = [class2id[predictions[i]['label']] for i in range(len(predictions))]

  labels = onehot(np.array(labels))
  preds = onehot(np.array(preds))

  return compute_metrics(preds, labels)
```

نمونه اجرای این تابع:

جمله ها و لیبل های نوشته شده در فایل(لیبل جمله سوم و چهارم که در تصویر نیفتاده به ترتیب sad و other است):

evaluate_test.csv

```
1  خیلی کوچیک هستن و سایزشون بدرد نمیخوره میخوام پس بدم    SAD
2  از صدای پرنده دم دمای صبح متنفرم متنفرم متنفرم    HATE
3  " ""بندگان محترم مجلس و دستگاه قضاء دیگه به این راحتی باخبر نشیم""
4  ؟؟ و اصن تاثیر داره؟ کسی اگه اطلاعی داره ممنون میشم راهنمایی کنید
5  این وضع ب طرز خنده داري گریه داره ...    SAD
```

نمونه خروجی تابع:

```
evaluate('evaluate_test.csv', '\t', classifier)
```

```
{'accuracy': 1.0,
 'precision_macro': 0.42857142857142855,
 'recall_macro': 0.42857142857142855,
 'f1_macro': 0.42857142857142855,
 'f1_C0': 0.0,
 'f1_C1': 1.0,
 'f1_C2': 0.0,
 'f1_C3': 0.0,
 'f1_C4': 0.0,
 'f1_C5': 1.0,
 'f1_C6': 1.0,
 'recall_C0': 0.0,
 'recall_C1': 1.0,
 'recall_C2': 0.0,
 'recall_C3': 0.0,
 'recall_C4': 0.0,
 'recall_C5': 1.0,
 'recall_C6': 1.0,
 'precision_C0': 0.0,
 'precision_C1': 1.0,
 'precision_C2': 0.0,
 'precision_C3': 0.0,
 'precision_C4': 0.0,
 'precision_C5': 1.0,
 'precision_C6': 1.0}
```

میبنیم که تمام متریک های گفته شده خروجی داده میشوند. علت صفر بودن برخی متریک ها این است که داده های نمونه هیچ مثالی از آن کلاس نداشته اند که متریک برایشان محاسبه شود.