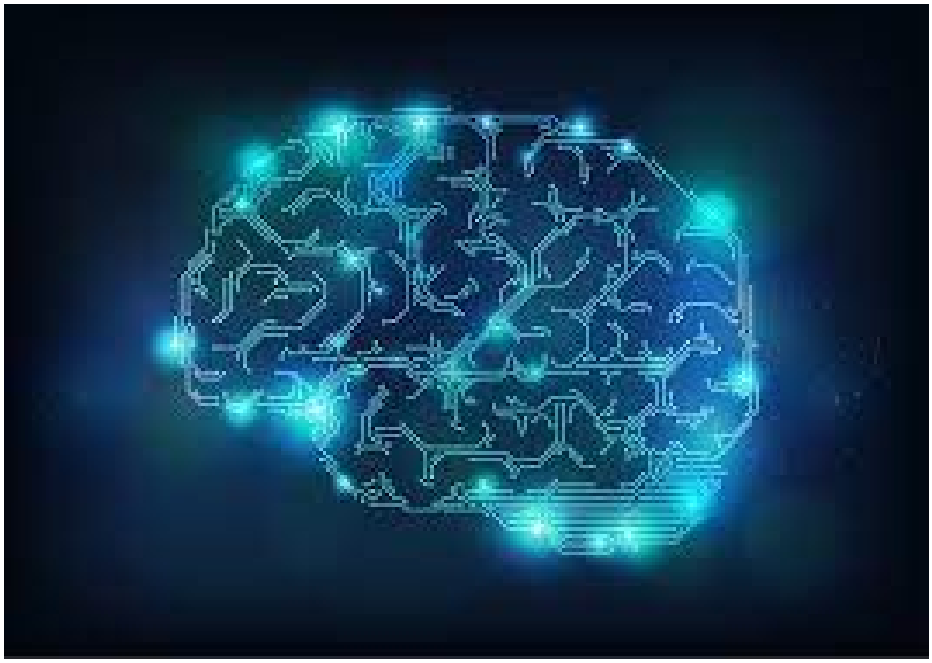
بسم الله الرحمن الرحیم



پارسا نوروزی 98411432

عرفان زارع 98522103

پروژه درس:

تحلیل احساسات

مدرس درس: استاد داوود آبادی

زمستان 1402

0.شرح موضوع:

تحلیل احساسات از مباحث مهم و داغ در صنایع هوش مصنوعی است.
با توجه به ترند بودن شبکه های اجتماعی واستفاده مردم از این برنامه ها ، شبکه های اجتماعی ناخودآگاه منابعی برای شناخت و بررسی افراد و شناخت آن ها می باشد. امروزه بسیاری از شرکت ها و دولت ها به شناخت و بررسی رفتار و علایق مردم و برخوردشان در بازه های زمانی متفاوت و مهم هستند تا با تحلیل آن بتوانند روند و رفتاری متناسب با آن را انجام بدهند.به همین سبب بررسی وتحلیل احساسات پیرامون افراد و متن های منتشر شده شان در شبکه اجتماعی می تواند عامل خوبی در تحلیل شخصیت و شناخت افراد باشد . به همین سبب مبحث sentiment analysis به مبحثی ترند و جدید در میان برنامه نویس ها و شرکت های مربوطه تبدیل شده است. در این پروژه نیز تلاش بر این شده که پروژه کوچکی از این بخش پیاده سازی شود.

1. شرح مجموعه دادگان

این دیتاست شامل بیش از 7000 جمله فارسی داخل توئیتر، اینستاگرام و دیجیکالا می باشد که در 7 کلاس شامل شادی، غم، عصبانیت، ترس،شگفتی و تنفر و دیگر، دسته بندی شده است. همچنین این دیتاسیت مربوط به مجموعه آرمان رایان شریف می باشد و توسط آنها جمع آوری شده است.

2. شرح روش های پیش پردازش

در راستای پیاده سازی فرایند پیش پردازش اقدامات انجام شده به شرح زیر میباشد:
ابتدا فایل های دیتاست را از آدرس زیر clone کردیم. (دو فایل train.tsv و test.tsv هر کدام به صورت جداگانه):

```
https://github.com/Arman-Rayan-Sharif/arman-text-emotion.git
```

سپس فایل ها را هر کدام در مسیر مناسب قرار دادیم. بعد هر کدام از فایل ها را به صورت یک دیتافریم خوانده و برای هر کدام از دیتا فریم های train و test را به text ها label ها تقسیم میکنیم. سپس به کمک دیکشنری label_dict هر کدام از label ها را به کد معادل آن تبدیل کردیم. (هم برای train و هم برای test).
سپس برای جلوگیری از حضور داده های نامربوط با کمک تابع fillna تمامی مقادیر خارج از حالت صحیح را جذف کردیم.همچنین با استفاده از کتابخانه هضم ، داده هارا نرمالایز نمودیم تا موارد اضافه حذف شود(موارد مدنظر در غکس داک هضم هست) . و نهایتا تمام داده ها را به صورت یک nested dictionary در متغیر data می گذاریم.

`Normalizer`

این کلاس شامل توابعی برای نرمال‌سازی متن است.

پارامترها:

نام	نوع	توضیحات	پیش‌فرض
`correct_spacing`	`bool`	اگر `True` فاصله‌گذاری‌ها را در متن، نشانه‌های سجاوندی و پیشوندها و پسوندها اصلاح می‌کند.	`True`
`remove_diacritics`	`bool`	اگر `True` باشد اعرابِ حروف را حذف می‌کند.	`True`
`remove_specials_chars`	`bool`	اگر `True` باشد برخی از کاراکترها و نشانه‌های خاص را که کاربردی در پردازش متن ندارند حذف می‌کند.	`True`
`decrease_repeated_chars`	`bool`	اگر `True` باشد تکرارهای بیش از ۲ بار را به ۲ بار کاهش می‌دهد. مثلاً «سلاممم» را به «سلامم» تبدیل می‌کند.	`True`
`persian_style`	`bool`	اگر `True` باشد اصلاحات مخصوص زبان فارسی را انجام می‌دهد؛ مثلاً جایگزین‌کردن کوتیشن با گیومه.	`True`
`persian_numbers`	`bool`	اگر `True` باشد ارقام انگلیسی را با فارسی جایگزین می‌کند.	`True`
`unicodes_replacement`	`bool`	اگر `True` باشد برخی از کاراکترهای یونیکد را با معادل نرمال‌شدهٔ آن جایگزین می‌کند.	`True`
`seperate_mi`	`bool`	اگر `True` باشد پیشوند «می» و «نمی» را در افعال جدا می‌کند.	`True`

دیکشنری label_dict:

```
label_dict = {
  'OTHER': 0,
  'HAPPY': 1,
  'SURPRISE': 2,
  'FEAR': 3,
  'HATE': 4,
  'ANGRY': 5,
  'SAD': 6,
}
```

کد انجام فرایند پیش پردازش:

```
from hazm import Normalizer
normalizer = Normalizer(persian_numbers=False,persian_style=False)
```

```
train_df = pd.read_table(f'{DATA_PATH}/train.tsv', header=None)
train_df[1] = train_df[1].map(label_dict)
train_texts, train_labels = train_df[0], train_df[1]
test_df = pd.read_table(f'{DATA_PATH}/test.tsv', header=None)
test_df[1] = test_df[1].map(label_dict)
test_texts, test_labels = test_df[0], test_df[1]
train_labels.fillna(0, inplace=True)
test_labels.fillna(0, inplace=True)
train_texts=train_texts.map(normalizer.normalize)
test_texts=test_texts.map(normalizer.normalize)
data = {
    'train': {'texts': train_texts, 'labels': train_labels},
    'test': {'texts': test_texts, 'labels': test_labels},
}
```

3. توضیح مدل های موجود

مدل های مناسب برای این تسک که یا برای زبان فارسی شخصی سازی شده اند یا به صورت چندزبانه(multi language) می باشد، شامل bert ،

parsbert ،roberta می باشد. در بررسی های انجام شده، مدل زبانی roberta، نتیجه بهتری نسبت به مدل parssbert دارد . همچنین در مقالات مطالعه شده نیز چنین چیزی به چشم میخورد .به علاوه مدل roberta به مراتب برای تسک های تحلیل احساسات، مناسب تر از bert است. در مقاله اصلی دیتاست جمع آوری شده نیز بدان اشاره شده است که مدلی که براساس دیتاست از توئیتر پیش آموزش دیده و چند زبانه است، به مراتب بهتر از مدل roberta ساده می باشد. بدین سان از میان مدل های مناسب موجود که مورد بررسی انجام گرفته ، با توجه به نتایج مورد بررسی قرار گرفته در مقالات و نتایج مشاهده شده در آمورش اولیه ، مدل xlm-roberta-large نتایج به مراتب بهتری داشته است و به عنوان پایه بررسی ما قرار گرفته است. همانطور که در تصویر زیر مشخص است، نتایج بدست آمده ،طبق بررسی در مقاله به شرح زیر است که به وضوح به برتری مدل انتخالی اشاره دارد.

Model	Precision (Macro)	Recall (Macro)	F1 (Macro)
FastText [42]	54.82	46.37	47.24
HAN [43]	49.56	44.12	45.10
RCNN [44]	50.53	48.11	47.95
RCNNVariant	51.96	48.96	49.17
TextAttBiRNN [45, 46]	54.66	46.26	47.09
TextBiRNN	51.45	47.16	47.14
TextCNN [47]	58.66	51.09	51.47
TextRNN [48]	49.39	47.20	46.79
ParsBERT	67.10	65.56	65.74
XLM-Roberta-base	72.26	68.43	69.21
XLM-Roberta-large	**75.91**	**75.84**	**75.39**
XLM-EMO-t	70.05	68.08	68.57

4. توضیح مدل انتخابی

مدل انتخابی برای پیاده سازی این مسئله مدل XLMRobertaModel بود.

این مدل یکی از انواع مدل های RoBERTa میباشد که روی داده های چندزبانه از پیش آموزش دیده است. این مدل برای هندل کردن چندین زبان طراحی شده است و روی حجم زیادی از داده های متنی به چندین زبان آموزش دیده است.معماری اصلی XLMRoberta بر اساس همان RoBERTa است که خودش یک مدل بر پایه transformer است. و از مکانیزم توجه به خود (self_attention machanism) برای فهمیدن روابط کلمات داخل یک جمله استفاده میکند. این مدل 100 زبان را پوشش داده و روی 2.5 میلیارد داده ی متنی آموزش داده شده است.
در راستای انجام امور مورد نیاز در این پروژه از LMRobertaClassifier , XLMRobertaTokenizer و XLMRobertaModel استفاده شده است.

5. توضیح مقاله مجموعه دادگان

مقاله مورد بررسی، به موضوع تشخیص احساسات از متن میپردازد و با توجه به افزایش دستزسی به شبکه های اجتماعی و داده متنی،توجه بسیاری از شرکت هارو به سمت خود برده است.بدین سان که با تحلیل احساسات مشتریان و کاربران سایت ها نسبت به اتفاقات، محصولات و خدمات به نتیجه و تصمیم مناسب پیرامون شرکت و خدمات یا تصمیمات مدیریتی می رسند.
این مقاله شامل بیش از 7000 جمله فارسی جمع آوری شده از نظرات ایسنتاگرام، توئیتر و نظرات پیرامون محصولات دیجی کالا می باشد که با 6 احساساس شامل خشم، شادی ،غم، عصبانیت، نفرت، حیرت می باشد .همچنین اگر شامل هیچ کدام از این احساسات نبود در دسته ، دیگران قرار می گیرد.
همچنین چندین مدل به عنوان مبنا برای طبقه بندی احساسات و این تسک ، مورد بررسی قرار گرفته است که بر روی مدل های مبتنی بر ترانسفورمر، که در زمینه nlp کاربرد دارد و عملکرد بهتری دارد، مورد بررسی قرار گرفته است که شامل BERT (Bidirectional Encoder Representations from Transformers) و DistilBERT است، این مدل ها با استفاده از یادگیری

عمیق، احساسات موجود در متن را تشخیص می دهد. همچنین در بررسی ابتدا داده هایشان را پیش پردازش می کنند تا برخی مشکلات نگارشی و غلط های املائی یا کلمات غیر فارسی ، حذف شوند. که این کار سبب تمیزتر شدن دیتاست و بهینه و مفید بودن داده های موجود می باشد. همچنین از برخی مدل ها که در قسمت های قبل بدان اشاره شده است استفاده شد که مدل بهینه ومناسب بدست آید و در نهایت مدل مدنظر یافت شد و در محاسبه وآموزش نهایی و محاسبه دقت و معیارها مورد ارزیابی قرار گرفت. که در این مقاله معیارها به شرح و فرمت زیر بدست آمده است:

Emotion	Precision	Recall	F1	Support (No. of Test Examples)
Anger	74.62	62.99	68.31	154
Fear	78.69	84.21	81.36	57
Happiness	86.02	87.27	86.64	275
Hatred	63.64	75.38	69.01	65
Other	60.98	77.72	68.34	193
Sadness	82.45	77.10	79.68	262
Wonder	84.96	66.21	74.42	145
Macro Average	75.91	75.84	75.39	1151

6. توضیح مقالات استفاده شده

در راستای انجام این پروژه از انواع سایت های مختلفی کمک گرفته شد که از مهم ترین آنها میتوان به stack_overflow, github و chatgpt اشاره کرد. شرح دقیق تر منابع استفاده شده در زیر آمده است:

https://poe.com/chat/1zoio5lmxy39rwvhxc8
https://huggingface.co/docs/transformers/model_doc/xlm-roberta
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

https://www.geeksforgeeks.org/different-ways-to-create-pandas-dataframe/
https://realpython.com/python-enumerate/
https://stackoverflow.com/questions/19929626/init-missing-1-required-positional-argument
https://chat.openai.com/c/8c983c2b-c598-46c1-b91f-5d6da3a14cb5
https://chat.openai.com/
https://pytorch.org/docs/stable/generated/torch.zeros.html
https://pytorch.org/docs/stable/generated/torch.no_grad.html
https://arodes.hes-so.ch/record/4525
https://ieeexplore.ieee.org/abstract/document/8791893
https://stackoverflow.com/questions/66389707/why-embed-dimemsion-must-be-divisible-by-num-of-heads-in-multiheadattention
https://stackoverflow.com/questions/66389707/why-embed-dimemsion-must-be-divisible-by-num-of-heads-in-multiheadattention
https://www.roshan-ai.ir/hazm/docs/
https://www.roshan-ai.ir/hazm/docs/content/hazm/normalizer.html
https://pandas.pydata.org/docs/reference/api/pandas.Series.map.html

7. توضیح پیاده سازی و اقدامات انجام شده

ابتدا درایو را روی سیستم mount می کنیم.

```
from google.colab import drive
drive.mount('/content/drive')
```

سپس داده را روی کولب کلون می نماییم و آدرس آن را ذخیره می کنیم.

```
from hazm import Normalizer
normalizer = Normalizer(persian_numbers=False,persian_style=False)
def preproc(text):
  ans=normalizer.normalize(text)
  return ans
```

سپس کتابخانه هضم را نصب می نماییم.

سپس کتابخانه های مود نیاز را نصب ادد می کنیم.

همچنین توکنایز و مدل ها و بهینه ساز های مدنظر را ادد می کند.همچنین لیبل هارا اضافه می کنیم.

همچنین برای classifier متن دیتاست نیز تابع زیر را می نویسیم. که توکنایزر و حداکثر طول آن نیز مقداردهی می شود. همچنین انکودینگ هم انجام شوند . همچنین لایه توجه و توکنایز کردن انجام شده و لیبل آن نیز ذخیره می شود.

```
class TextClassificationDataset(Dataset):
  def __init__(self, texts, labels, tokenizer, max_length):
    self.texts = texts
    self.labels = labels
    self.tokenizer = tokenizer
    self.max_length = max_length

  def __len__(self):
    return len(self.texts)

  def __getitem__(self, idx):
    text = self.texts[idx]
    label = self.labels[idx]
    encoding = self.tokenizer(text, return_tensors='pt',
max_length=self.max_length, padding='max_length', truncation=True)
```

```
    return {'input_ids': encoding['input_ids'].flatten(),
'attention_mask': encoding['attention_mask'].flatten(), 'label':
torch.tensor(label)}
```

بخش پیش پردازش هم در قسمت مروبطه توضیح داده شده بود. حال کلاس کلاسیفایر هر مدل را تعریف می کنیم. که بخش پردازش مدل ها و دراپ اوت و لایه خطی مد نظر را پیاده سازی و پردازش می کنیم . تابع فوروارد آن پیاده می شود که شامل لایه پولینگ و دراپ اوت و لایه خطی است که روی ورودی با مدل bert زده شده(ورودی لایه ها و لایه توجه داده می شود)

```
class XLMRobertaClassifier(nn.Module):
  def __init__(self, model_name, num_classes):
    super(XLMRobertaClassifier, self).__init__()
    self.bert = XLMRobertaModel.from_pretrained(model_name)
    self.dropout = nn.Dropout(0.1)
    self.fc = nn.Linear(self.bert.config.hidden_size, num_classes)

  def forward(self, input_ids, attention_mask):
    outputs = self.bert(input_ids=input_ids,
attention_mask=attention_mask)
    pooled_output = outputs.pooler_output
    x = self.dropout(pooled_output)
    logits = self.fc(x)
    return logits
```

در ادامه به تابع train میرسیم که با گرفتن پارامتر های model, data_laoder, optimizer, scheduler, و device مدل را روی داده های data_loader آموزش میدهد. به این ترتیب که اول تابع train را روی مدل صدا زده و بعد روی داده ها یک حلقه زده شده و در هر گام مراحل زیر انجام میشود:

ست کردن گرادیان و سپس انجام پیش بینی با دادن آیدی داده ها و ماسک توجه که در آخر رویشان یک فعالسازی softmax زده میشود. سپس محاسبه loss از روی crossEntropy. و بعد انجام backpropagatoin و اعمال optimizer. نهایتا بیشترین نمره ی پیش بینی را ذخیره کرده و به لیست پیش بینی ها در کنار اضافه شدن برچسب واقعی به لیست برچسب ها.
و در آخر بعد از تمام شدن حلقه متریک های دقت را برمیگرداند.(این متریک ها را به کمک فرستادن لیست پیش بینی ها و برچسب ها به دو تابع accuracy_score و classification_report انجام میدهد محاسبه میکند).

```
def train(model, data_loader, optimizer, scheduler, device):
  model.train()
  predictions = []
  actual_labels = []
  for batch in tqdm(data_loader, position=0):
    optimizer.zero_grad()
    input_ids = batch['input_ids'].to(device)
    attention_mask = batch['attention_mask'].to(device)
    labels = batch['label'].to(device)
    outputs = model(input_ids=input_ids, attention_mask=attention_mask)
    labels = labels.long()
    probabilities = F.softmax(outputs, dim=1)
    loss = nn.CrossEntropyLoss()(probabilities, labels)
    loss.backward()
    optimizer.step()
    scheduler.step()
    _, preds = torch.max(outputs, dim=1)
    predictions.extend(preds.cpu().tolist())
    actual_labels.extend(labels.cpu().tolist())
  return accuracy_score(actual_labels, predictions),
classification_report(actual_labels, predictions)
```

تابع evaluate هم تقریبا ساختاری شبیه به تابع train دارد به جز اینکه پس از دریافت خروجی پیش بینی ها، دیگر هیچ یک از مراحل: تابع فعالسازی، محاسبه

ضرر به کمک تابع ضرر، انجام back propagation و اعمال بهینه ساز را انجام نمیدهد.

```
def evaluate(model, data_loader, device):
  model.eval()
  predictions = []
  actual_labels = []
  with torch.no_grad():
    for batch in tqdm(data_loader, position=0):
      input_ids = batch['input_ids'].to(device)
      attention_mask = batch['attention_mask'].to(device)
      labels = batch['label'].to(device)
      outputs = model(input_ids=input_ids, attention_mask=attention_mask)
      _, preds = torch.max(outputs, dim=1)
      predictions.extend(preds.cpu().tolist())
      actual_labels.extend(labels.cpu().tolist())
  return accuracy_score(actual_labels, predictions),
classification_report(actual_labels, predictions)
```

در گام بعدی پارامتر های اصلی مدل و آموزش به طور دستی و پس از چندین بار تست کردن حالات مختلف مقدار دهی شدند:

```
num_classes = 7
max_length = 128
batch_size = 32
num_epochs = 6
learning_rate = 2e-5
```

در مرحله ی بعد تابع main پیاده سازی میشود که ورودی های زیر را میگیرد: داده ها، مدل انتخابی، لایه، toeknizer و classifier
ابتدا toeknizer از روی کلاس مدل ساخته میشود. بعد دیتاست های train و validation جداسازی و مرتب میشود. سپس در صورت امکان divice را روی gpu ست میکند. بعد برای همه لایه ها غیر از لایه ای که در ورودی گرفتیم گرادیان را false میکند. بعد بهینه ساز را ست میکند و روی epoch ها

حلقه میزند و در هر بار tarin و evaluate را انجام میدهد. و نهایتا گزارش ها و مدل را برمیگرداند.

```
def main(data, language_model, layer=None, tokenizer_class=AutoTokenizer, classifier_class=XLMRobertaClassifier):
    tokenizer = tokenizer_class.from_pretrained(language_model)
    train_dataset = TextClassificationDataset(data['train']['texts'], data['train']['labels'], tokenizer, max_length)
    val_dataset = TextClassificationDataset(data['test']['texts'], data['test']['labels'], tokenizer, max_length)
    train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    val_dataloader = DataLoader(val_dataset, batch_size=batch_size)

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = classifier_class(language_model, num_classes).to(device)

    if layer:
        for name, param in model.named_parameters():
            if layer in name:
                break
            param.requires_grad = False

    optimizer = torch.optim.AdamW(model.parameters(), lr=learning_rate)
    total_steps = len(train_dataloader) * num_epochs
    scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0, num_training_steps=total_steps)

    reports = {'train': list(), 'val': list()}
    for epoch in range(num_epochs):
        print(f"Epoch {epoch + 1}/{num_epochs}")
        train_accuracy, train_report = train(model, train_dataloader, optimizer, scheduler, device)
        val_accuracy, val_report = evaluate(model, val_dataloader, device)
        print(f"Train Accuracy: {train_accuracy:.4f}, Validation Accuracy: {val_accuracy:.4f}")
        print('-' * 20)
        reports['val'].append([val_accuracy, val_report])
        reports['train'].append([train_accuracy, train_report])
    print(val_report)
```

```
    return model, reports
```

تابع بعدی plot_report است که طبق گزارشات نمودار سیر دقت train و validation را در هر epch نشان میدهد.

```
def plot_report(reports):
  val = [i[0] for i in reports['val']]
  train = [i[0] for i in reports['train']]
  x = range(len(val))
  plt.plot(x, train, label='train')
  plt.plot(x, val, label='val')
  plt.legend()
```

بعد با ارسال مدل xlm_roberta_large تابع main را اجرا میکند.

```
xlm_roberta_model = 'xlm-roberta-large'
xlm_roberta_model_instance, xlm_roberta_reports = main(data,
xlm_roberta_model, layer='11', tokenizer_class=XLMRobertaTokenizer,
classifier_class=XLMRobertaClassifier)
```

بعد با صدا زدن تابع plot_report گزارشات را رسم میکند.

```
plot_report(xlm_roberta_reports)
```

و نهایتا مدل را ذخیره میکند:

```
import joblib

joblib.dump(xlm_roberta_model_instance,
'/content/drive/MyDrive/model_proj.joblib')

loaded_model = joblib.load('/content/drive/MyDrive/model_proj.joblib')
```

8. حداقل دقت بدست آمده

در طول اجرا های انجام شده با انواع پارامتر ها دقت از روی 17 شروع شده و با بررسی های مختلف نهایتا به دقت 71 درصد رسیدیم:

	precision	recall	f1-score	support
0	0.59	0.79	0.68	193
1	0.86	0.71	0.78	275
2	0.82	0.55	0.66	145
3	0.65	0.82	0.73	57
4	0.60	0.55	0.58	65
5	0.67	0.57	0.62	154
6	0.70	0.81	0.75	262
accuracy			0.71	1151
macro avg	0.70	0.69	0.68	1151
weighted avg	0.72	0.71	0.70	1151

9. ارزیابی مدل با انواع معیارها

با استفاده از ابرپارامتر های موجود و بررسی و تغییر به دنبال بهترین مقادیرر برای خروجی معیارها بودیم که در نهایت بر اساس آن به نتایج مدنظر نزدیک شدیم.ما براساس ابرپارامترهایی همچون batch_size، حداکثر طول متن، تعداد دورو نرخ آموزش می باشد. ما در محاسباتمان با استفاده از کتابخانه های آماده به محاسبه دقت پیش بینی و precision,recall,f1-score پرداختیم که براساس این معیار ها به نتایج زیر دست یافتیم:

نهایی(پیش پردازش):

```
[17] num_classes = 7
     max_length = 128
     batch_size = 32
     num_epochs = 6
     learning_rate = 2e-5
```

	precision	recall	f1-score	support
0	0.58	0.81	0.68	193
1	0.91	0.66	0.77	275
2	0.82	0.64	0.72	145
3	0.76	0.77	0.77	57
4	0.81	0.38	0.52	65
5	0.68	0.64	0.66	154
6	0.69	0.89	0.77	262
accuracy			0.72	1151
macro avg	0.75	0.68	0.70	1151
weighted avg	0.75	0.72	0.72	1151

موارد تست شده:

```
num_classes = 7
max_length = 128
batch_size = 32
num_epochs = 5
learning_rate = 2e-4
```

	precision	recall	f1-score	support
0	0.17	1.00	0.29	193
1	0.00	0.00	0.00	275
2	0.00	0.00	0.00	145
3	0.00	0.00	0.00	57
4	0.00	0.00	0.00	65
5	0.00	0.00	0.00	154
6	0.00	0.00	0.00	262
accuracy			0.17	1151
macro avg	0.02	0.14	0.04	1151
weighted avg	0.03	0.17	0.05	1151

```
num_classes = 7
max_length = 128
batch_size = 32
num_epochs = 5
learning_rate = 6e-5
```

	precision	recall	f1-score	support
0	0.17	1.00	0.29	193
1	0.00	0.00	0.00	275
2	0.00	0.00	0.00	145
3	0.00	0.00	0.00	57
4	0.00	0.00	0.00	65
5	0.00	0.00	0.00	154
6	0.00	0.00	0.00	262
accuracy			0.17	1151
macro avg	0.02	0.14	0.04	1151
weighted avg	0.03	0.17	0.05	1151

```
num_classes = 7
max_length = 128
batch_size = 16
num_epochs = 5
learning_rate = 2e-5
```

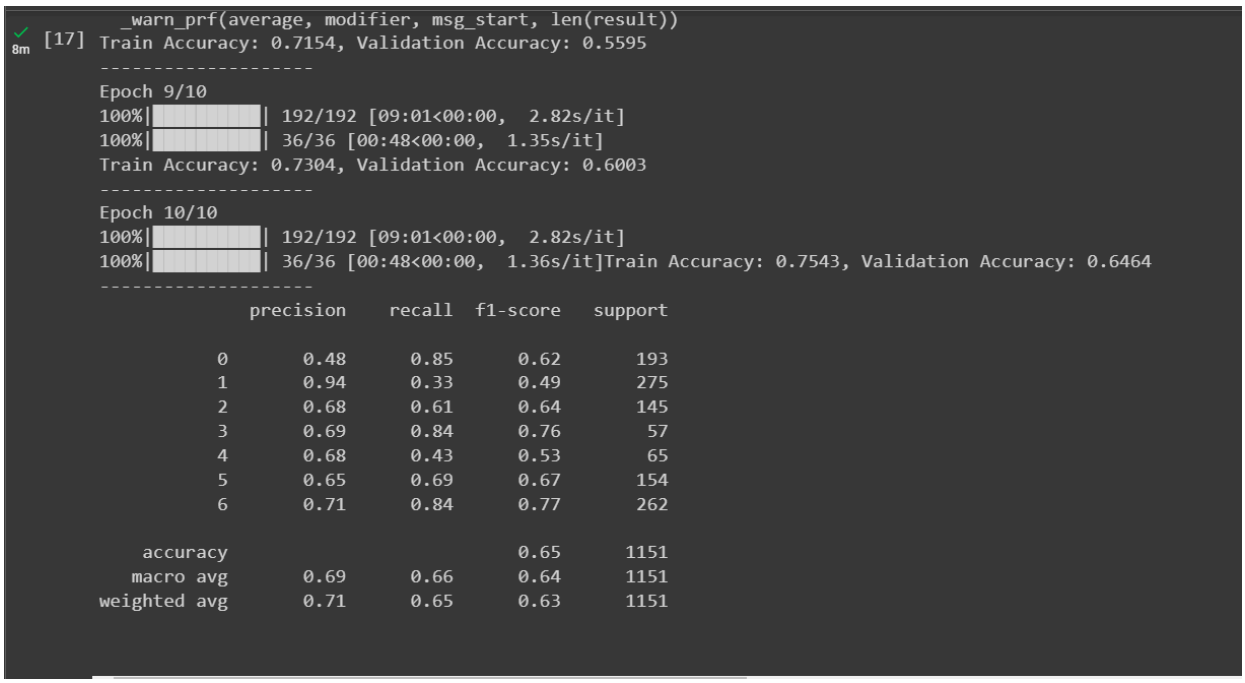
	precision	recall	f1-score	support
0	0.49	0.75	0.59	193
1	0.82	0.51	0.63	275
2	0.70	0.61	0.65	145
3	0.58	0.86	0.70	57
4	0.67	0.43	0.52	65
5	0.65	0.53	0.58	154
6	0.66	0.77	0.71	262
accuracy			0.64	1151
macro avg	0.65	0.64	0.63	1151
weighted avg	0.67	0.64	0.64	1151

```
num_classes = 7
max_length = 128
batch_size = 32
num_epochs = 5
learning_rate = 2e-5
```

	precision	recall	f1-score	support
0	0.55	0.76	0.64	193
1	0.88	0.63	0.73	275
2	0.70	0.64	0.67	145
3	0.69	0.82	0.75	57
4	0.70	0.35	0.47	65
5	0.68	0.63	0.65	154
6	0.69	0.82	0.75	262
accuracy			0.69	1151
macro avg	0.70	0.67	0.67	1151
weighted avg	0.71	0.69	0.69	1151

```
num_classes = 7
max_length = 256
```

```
batch_size = 32
num_epochs = 10
learning_rate = 2e-5
```



```
  _warn_prf(average, modifier, msg_start, len(result))
[17] Train Accuracy: 0.7154, Validation Accuracy: 0.5595
--------------------
Epoch 9/10
100%|██████████| 192/192 [09:01<00:00,  2.82s/it]
100%|██████████| 36/36 [00:48<00:00,  1.35s/it]
Train Accuracy: 0.7304, Validation Accuracy: 0.6003
--------------------
Epoch 10/10
100%|██████████| 192/192 [09:01<00:00,  2.82s/it]
100%|██████████| 36/36 [00:48<00:00,  1.36s/it]Train Accuracy: 0.7543, Validation Accuracy: 0.6464
--------------------
              precision    recall  f1-score   support

           0       0.48      0.85      0.62       193
           1       0.94      0.33      0.49       275
           2       0.68      0.61      0.64       145
           3       0.69      0.84      0.76        57
           4       0.68      0.43      0.53        65
           5       0.65      0.69      0.67       154
           6       0.71      0.84      0.77       262

    accuracy                           0.65      1151
   macro avg       0.69      0.66      0.64      1151
weighted avg       0.71      0.65      0.63      1151
```

با پیش پردازش با هضم:

```
num_classes = 7
max_length = 128
batch_size = 32
num_epochs = 5
learning_rate = 2e-5
```

	precision	recall	f1-score	support
0	0.60	0.85	0.70	193
1	0.91	0.74	0.82	275
2	0.78	0.63	0.70	145
3	0.69	0.81	0.74	57
4	0.72	0.40	0.51	65
5	0.71	0.58	0.64	154
6	0.72	0.83	0.77	262
accuracy			0.73	1151
macro avg	0.73	0.69	0.70	1151
weighted avg	0.75	0.73	0.73	1151

```
num_classes = 7
max_length = 128
batch_size = 32
num_epochs = 5
learning_rate = 2e-5
```

---

	precision	recall	f1-score	support
0	0.59	0.79	0.68	193
1	0.86	0.71	0.78	275
2	0.82	0.55	0.66	145
3	0.65	0.82	0.73	57
4	0.60	0.55	0.58	65
5	0.67	0.57	0.62	154
6	0.70	0.81	0.75	262
accuracy			0.71	1151
macro avg	0.70	0.69	0.68	1151
weighted avg	0.72	0.71	0.70	1151

10.گزارش کردن چند نمونه ورودی و خروجی مدل

با استفاده از تابع predict در اینجا چند نمونه از گزارشان ورودی را به همراه پیش بینی مدل در کنار label اصلی آن نمایش میدهیم:

HATE	HATE	واقعا بیدلیل ازش بدم نمیاد
ANGRY	ANGRY	هم خس و خاشاکن ؟ مردم هم بد بختن و شما از همه بهتر میبینید و میدانید! تریبون اصلی مملکت هم در اختیار شماس، چه قدمی برداشتید جز سانسور و جز شانتاژ رسانه ای؟
OTHER	OTHER	فکی و نمایشی هستند. مثلا اینها در موضوع شهرداری معتقدند شهردار باید کسی باشد که با همه بسازد و دستگاههای دیگر به آن اعتراض نداشته باشند/شرق #اصلاحطلبان_فکی
ANGRY	ANGRY	
HAPPY	HAPPY	
FEAR	FEAR	تمادی و خشم نسبت به حاکمیت واقعا ترسناکه

که در ردیف سمت راست نمونه متن ورودی آمده. در وسط label اصلی و در ردیف سمت چپ پیش بینی مدل از متن ورودی نمایش داده شده است.

11. آماده سازی توابع خواسته شده

طبق خواسته doc اصلی پروژه باید دو تابع predict برای پیش بینی نمونه های جدید، و evaluate برای محاسبه accuracy, f1_score, precision و recall نوشته میشد.
در راستای محقق شدن این امر در نهایت از 3 تابع استفاده شد:

Predict_emoji
Predict
Evaluate

تابع predict_emoji: یک تابع کمکی است که در اصل میتواند با گرفتن یک یا لیستی از داده ها، تخمین بزند که نوع sentiment هر یک از جملات متعلق به کدام یک از دسته های sentimental میباشد. ورودی هایی که این تابع میگیرد عبارتند از: لیست texts، مدل، tokenizer، دیکشنری ای که برای map کردن حالات به کد های آنها استفاده میشود و در نهایت max_length.
این تابع روی لیست متن ها یک iteration میزند و برای هر کدام با کمک ماسک attention و encoding input id ها فرایند پیش بینی را انجام داده و

در ادامه با استفاده از دو تابع فعالسازی sigmoid و softmax به آرایه ای از نمره ی نهایی برای هر کلاس را در مورد هر متن محاسبه میکند. سپس آیتمی که بیشترین نمره را دارد را به عنوان پیش بینی نهایی در نظر میگیرد و خود label را به همراه کد مربوط به آن به انتهای دو لیست predicts و predict_codes اضافه میکند. و نهایتا این دو لیست را به عنوان خروجی های نهایی برمیگرداند:

```
def predict_emoji(texts, model, tokenizer= Tokenizer, label_dict=
reversed_label_Dict, max_length=128):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
## if you can use cuda(with gpu) use it, otherwise use cpu
    model.eval()
    num_classes = len(label_dict)
    preds = torch.zeros(num_classes).to(device)
    linear = torch.nn.Linear(model.fc.in_features, num_classes).to(device)
    predicts = []
    predict_codes = []

    for text in tqdm(texts,position=0):

        encoding = tokenizer(text, return_tensors='pt',
max_length=max_length, padding='max_length', truncation=True)
        input_ids = encoding['input_ids'].to(device)
        attention_mask = encoding['attention_mask'].to(device)

        with torch.no_grad():
            logits = model(input_ids=input_ids,
attention_mask=attention_mask)
            preds += torch.sigmoid(logits).sum(dim=0)

        preds /= len(texts)
        preds = F.softmax(preds, dim=0)
        _, label_idx = torch.max(preds, dim=0)
        label = label_dict[label_idx.item()]
        predicts.append(label)
        predict_codes.append(label_idx.item())

    return predicts , predict_codes
```

تابع predict: این تابع وظیفه دارد تا با گرفتن مسیر فایل داده های تست جدید و ایندکس های شروع و پایان از آیتم های فایل مذکور، پیش بینی را برای متن های انتخاب شده انجام داده و پس از چاپ جدول داده-پیش بینی- برچسب، دو لیست پیش بینی ها و کد های مربوط به آنها را برگرداند.
ورودی های این تابع عبارتند از: آدرس فایلی که در آن داده های مورد نظر ذخیره شده اند. ایندکس شروع و ایندکس پایانی که قرار است از فایل خوانده شود.(که البته این دو پارامتر اختیاری هستند و در صورت ندادن به صورت پیش فرض تمام لیست در نظر گرفته میشود)
این تابع فایل ورودی را به عنوان یک دیتافریم میخواند، برچسب ها را با map کردن از روی label_dict به کد های معادل آنها تبدیل کرده و در کنار متن ها در متغیر data به شکل یک دیکشنری نگه میدارد. سپس با فراخوانی تابع predict_emoji دو لیست پیش بینی ها و کد های آن را بدست آورده و در کنار برچسب های اصلی و متن های مربوط به هر یک چاپ میکند. و در نهایت دو لیست predicts و predict_codes را برمیگرداند:

```
def predict(path, from_idx =0 , to_idx=-1):
    test_df = pd.read_table(path, header=None)
    # test_df = pd.read_csv(path, header=None)
    test_df[1] = test_df[1].map(label_dict)
    test_texts, test_labels = test_df[0], test_df[1]
    data = {'texts': test_texts, 'labels': test_labels,}
    if to_idx ==-1:
      predicts, predict_codes =
predict_emoji(texts=data['texts'].tolist()[from_idx:], model=loaded_model)
    else:
      predicts, predict_codes =
predict_emoji(texts=data['texts'].tolist()[from_idx:to_idx],
model=loaded_model)

    temp = []
    for i, predict in enumerate(predicts):
      # print(i,', predict: ', predict,', label: ',
reversed_label_Dict[data['labels'][i]],', text: ', data['texts'][i])
```

```
temp.append([predict,reversed_label_Dict[data['labels'][i]],data['texts'][
i]])

    df = pd.DataFrame(temp, columns=['Predict', 'Label', 'Text'])
    table = df.to_string()
    print()
    print(table)

    return predicts, predict_codes
```

تابع evaluate:

این تابع هم دقیقا با دریافت ورودی های مشابه با تابع predict، به کمک تابع predict_emoji لیست پیش بینی ها را میگیرد.

تنها تفاوت اصلی این تابع با تابع predict در اینجاست که نمونه های پیش بینی شده را چاپ نکرده و علاوه بر این به کمک کتابخانه های accuracy_score و classification_report مقادیر accuracy, f1_score, precision و recall را محاسبه میکند. در نهایت هم این مقادیر را چاپ کرده و برمیگرداند:

```
def evaluate(path, from_idx =0 , to_idx=-1):
    test_df = pd.read_table(path, header=None)
    test_df[1] = test_df[1].map(label_dict)
    test_texts, test_labels = test_df[0], test_df[1]
    data = {'texts': test_texts, 'labels': test_labels,}
    if to_idx ==-1:
      predicts, predict_codes =
predict_emoji(texts=data['texts'].tolist()[from_idx:], model=loaded_model)
      score = accuracy_score(data['labels'].tolist()[from_idx:],
predict_codes)
      report = classification_report(data['labels'].tolist()[from_idx:],
predict_codes)
    else:
      predicts, predict_codes =
predict_emoji(texts=data['texts'].tolist()[from_idx:to_idx],
model=loaded_model)
```

```
      score = accuracy_score(data['labels'].tolist()[from_idx:to_idx],
predict_codes)
      report =
classification_report(data['labels'].tolist()[from_idx:to_idx],
predict_codes)
    print()
    print('accuracy: ', score)
    print(report)
    return score, report
```

منابع:

مقاله اصلی:

https://arxiv.org/pdf/2207.11808.pdf

https://poe.com/chat/1zoio5lmxy39rwvhxc8

https://huggingface.co/docs/transformers/model_doc/xlm-roberta

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

https://www.geeksforgeeks.org/different-ways-to-create-pandas-dataframe/

https://realpython.com/python-enumerate/

https://stackoverflow.com/questions/19929626/init-missing-1-required-positional-argument

https://chat.openai.com/c/8c983c2b-c598-46c1-b91f-5d6da3a14cb5

https://chat.openai.com/

https://pytorch.org/docs/stable/generated/torch.zeros.html

https://pytorch.org/docs/stable/generated/torch.no_grad.html
https://arodes.hes-so.ch/record/4525
https://ieeexplore.ieee.org/abstract/document/8791893
https://stackoverflow.com/questions/66389707/why-embed-dimemsion-must-be-divisible-by-num-of-heads-in-multiheadattention
https://stackoverflow.com/questions/66389707/why-embed-dimemsion-must-be-divisible-by-num-of-heads-in-multiheadattention
https://huggingface.co/HooshvareLab/bert-base-parsbert-uncased?library=true
https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html
https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html
https://huggingface.co/bert-base-uncased
https://arxiv.org/abs/2005.12515