گزارش تمرین چهارم درس پردازش زبان طبیعی

تیم شماره ۱۰

تسک تحلیل احساسات از روی اخبار فارسی

قسمت اول: استفاده از مدل از پیش آموزش داده‌شده برای دسته‌بندی متون اخبار فارسی به جهت تحلیل احساسات

در این قسمت ابتدا با استفاده از آدرس مشخص شده داده‌ها دانلود شده و با استفاده از کتابخانه json پایتون که یک internal library است داده ها تبدیل می‌شوند.

کتابخانه های مورد استفاده موارد زیر هستند که کاربرد هر یک توضیح داده خواهد شد.

```
!pip install hazm -Uqq
!pip install transformers -Uqq
!pip install pytorch-lightning -Uqq
!pip install torchmetrics
!pip install lightning-transformers
```

کتابخانه هضم برای پیش پردازش متن فارسی مورد استفاده قرار گرفته.
کتابخانه ترنفسورمرز که برای لودکردن مدل فارسی و استفاده از ان استفاده شده است.
کتابخانه مهم بعدی کتابخانه تورچ لایتینیگ هست که یک wrapper برای پایتورچ هست و عملا در حال تبدیل شدن به یک استاندارد مهم است و کتابخانه torchmetrics نیز برای محاسبات معیارهای ماشین لرنینگ استفاده شده است.

```python
from transformers import pipeline
from transformers import BertTokenizer, BertModel

model_name_or_path = "HooshvareLab/bert-fa-zwnj-base"

tokenizer = BertTokenizer.from_pretrained(model_name_or_path)
```

مدل استفاده شده در اینجا مدل پارس برت هست که از توکنایزر و مدل پارس برت برای تنظیم دقیق داده استفاده شده است.

```python
[9] import json
    from pathlib import Path
    dataset = json.loads(Path('dataset_annotated_sentiment.json').read_text())

    train_texts, train_labels = read_data_convert(dataset['train'],voting=True)
    val_texts, val_labels = read_data_convert(dataset['eval'],voting=True)
    test_texts, test_labels = read_data_convert(dataset['test'],voting=True)

    train_encodings = tokenizer(train_texts, truncation=True, padding=True,max_length=256)
    val_encodings = tokenizer(val_texts, truncation=True, padding=True,max_length=256)
    test_encodings = tokenizer(test_texts, truncation=True, padding=True,max_length=256)
```

سپس با استفاده از تابع read_data_convert مد دیتا بدست آمده است. طبق تجربه اینجانب استفاده از همه داده ها مناسب نبوده و مدل خوب یاد نمیگیرد به همین جهت بنده از مد لیبل ها استفاده نمودم.

سپس با استفاده از توکنایزر مدل برت کلمات به انکدینگ مناسب تبدیل می‌شوند.

```python
import torch

class SentimentDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

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

train_dataset = SentimentDataset(train_encodings, train_labels)
val_dataset = SentimentDataset(val_encodings, val_labels)
test_dataset = SentimentDataset(test_encodings, test_labels)
```

برای تبدیل دیتا به فرمت مناسب دیتالودر از ماژول دیتاست پایتورچ ارث بری شده است و استفاده گردیده است.

```python
import torch.nn as nn
from transformers import AdamW, get_linear_schedule_with_warmup
from torchmetrics import AUROC

BERT_MODEL_NAME = "HooshvareLab/bert-fa-zwnj-base"


from transformers import TrainingArguments, Trainer,AutoModelForSequenceClassification
from torch.utils.data import DataLoader

model = AutoModelForSequenceClassification.from_pretrained(BERT_MODEL_NAME,
                                                           num_labels=3,
                                                           label2id=label2id,
                                                           id2label=id2label)
model.config
--NORMAL--
```

سپس مدل برت لود می‌شود.

```python
from pytorch_lightning import Trainer
from pytorch_lightning.callbacks import ModelCheckpoint


checkpoint_callback = ModelCheckpoint(dirpath='/content/models')

BATCH_SIZE = 16
net = LiteModel(model, train_dataset, val_dataset, test_dataset, BATCH_SIZE)
trainer = Trainer(accelerator='auto',
                  devices='auto',
                  max_epochs=10,
                  #overfit_batches=1,
                  log_every_n_steps=1,
                  callbacks=[checkpoint_callback])
```

در استفاده از ماژول لایتینگ چند نکته مهم وجود دارد. یکی سیو کردن مدل‌ها با استفاده از checkpoint است و دیگری overfit_batch که برای این استفاده می‌شود که مدل بتواند روی یک بچ اورفیت کند و مطمئن شویم مدل درست کار می‌کند!

تابع init در مدل بصورت زیر تعریف شده است و سه معیار accuracy و f1score و auroc را محاسبه می‌کند.

```python
class LiteModel(pl.LightningModule):
    def __init__(self,model, train_dataset, val_dataset, test_dataset, batch_size):
        super(LiteModel, self).__init__()
        self.model = model
        self.train_dataset = train_dataset
        self.val_dataset = val_dataset
        self.test_dataset = test_dataset
        self.batch_size = batch_size

        self.train_acc = torchmetrics.Accuracy()
        self.train_f1 = torchmetrics.F1Score(num_classes=3)
        self.train_auroc = torchmetrics.AUROC(num_classes=3)

        self.val_acc = torchmetrics.Accuracy()
        self.val_f1 = torchmetrics.F1Score(num_classes=3)
        self.val_auroc = torchmetrics.AUROC(num_classes=3)

        self.test_acc = torchmetrics.Accuracy()
        self.test_f1= torchmetrics.F1Score(num_classes=3)
        self.test_auroc = torchmetrics.AUROC(num_classes=3)
```

سپس در اینجا برای تعریف training step که در واقع روش backpropagation روی هر بچ است از تابع زیر استفاده می‌شود.

```
def training_step(self, batch, batch_idx):
    output= self(batch)

    softmax = output.logits.softmax(dim=1)
    y_pred = torch.argmax(softmax, dim=1)
    y_target = batch['labels']
    acc = self.train_acc(y_pred, y_target)
    f1 = self.train_f1(y_pred, y_target)
    self.train_auroc.update(softmax, y_target)

    self.log("train_loss", output.loss,prog_bar=True)
    self.log("train_accuracy", acc,prog_bar=True)
    self.log("train_f1", f1,prog_bar=True)

    return output.loss
```

برای val هم به همین صورت کار انجام می‌شود.

و برای ترین کردن مدل از تابع trainer.fit استفاده کردیم که نتایج آن در فایل ژوپیتر بصورت کامل مشخص است و در نهایت تست مدل با استفاده از بالانس سازی کلاس ها به شکل زیر است.

```
test accuracy: 0.5676 f1: 0.5676, auroc: 0.5
```

Test metric	DataLoader 0
test_accuracy	0.5675675868988037
test_auroc	0.5
test_f1	0.5675675868988037
test_loss	0.948407769203186

قسمت دوم: استفاده از یک شبکه LSTM برای دسته‌بندی متون اخبار فارسی به جهت تحلیل احساسات

فاز اول: پیش پردازش داده‌ها

در این بخش ابتدا از آدرس مشخص شده، داده‌هارا دانلود کرده و پس از تبدیل داده‌ها به فرمت JSON با استفاده از کتابخانه پایتون HAZM، تمامی متون را Normalize و Lemmatize می‌کنیم. هم‌چنین برچسب داده‌هارا به به شکل زیر تبدیل می‌کنیم:

اخبار منفی: برچسب 1-

اخبار خنثی: برچسب 0

اخبار مثبت: برچسب 1

```
'''
normalize and lemmatize train & eval dataset and convert to list format, and also
convert labels to numerical format
'''

augmented_train = []
# this is correct. But it shows bad because of RTL
labels = {'1- :'منفی' ,1 :'مثبت' ,0 :'خنثی'}
for item in train:
    for label in item['annotations']:
        augmented_train.append((lemmatizer.lemmatize(normalizer.normalize(item['text'])),labels[label]))
for item in eval:
    for label in item['annotations']:
        augmented_train.append((lemmatizer.lemmatize(normalizer.normalize(item['text'])),labels[label]))
```

```
'''
normalize and lemmatize test dataset and convert to list format, and also
convert labels to numerical format
'''

augmented_test = []
# this is correct. But it shows bad because of RTL
labels = {'1- :'منفی' ,1 :'مثبت' ,0 :'خنثی'}
for item in test:
    for label in item['annotations']:
        augmented_test.append((lemmatizer.lemmatize(normalizer.normalize(item['text'])),labels[label]))
```

سپس به جهت اینکه آسان‌تر بتوانیم با داده‌ها کار کنیم، آن‌هارا تبدیل به دیتافریم‌های کتابخانه PANDAS کرده و به وسیله تابع زیر، تمامی ۱-حروف انگلیسی ۲-اعداد ۳-کاراکترهای خاص مانند نیم‌فاصله و کاراکتر Newline ۴-گیومه فارسی ۵-کاراکترهای نقطه‌گذاری ۶-آدرس‌های اینترنتی را به روش عبارت‌های منظم، حذف می‌کنیم.

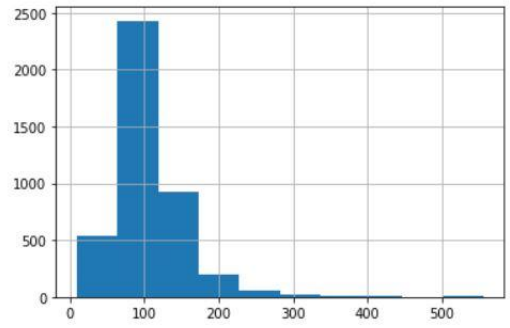
```
def remove_english_number(text):
  text = re.sub("[a-zA-Z0-9\u200c\n\t_»«]+", "",text)
  text = re.sub("[^\w\s]",'',text)
  text = re.sub('https://.*','',text)
  return text
```

در مرحله بعد، به کمک کتابخانه Counter یک دیکشنری به شکل (کلمه،ایندکس) از تمام کلمات درون متون استخراج کرده تا در آینده بتوانیم کلمات درون متون را به یک عدد طبیعی تبدیل کنیم.

```
all_text = ' '.join(augmented_train[0])
all_text = all_text + ' '.join(augmented_test[0])
# create a list of words
words = all_text.split()
# Count all the words using Counter Method
count_words = Counter(words)
total_words = len(words)
sorted_words = count_words.most_common(total_words)
vocab_to_int = {w:i+1 for i, (w,c) in enumerate(sorted_words)}
```

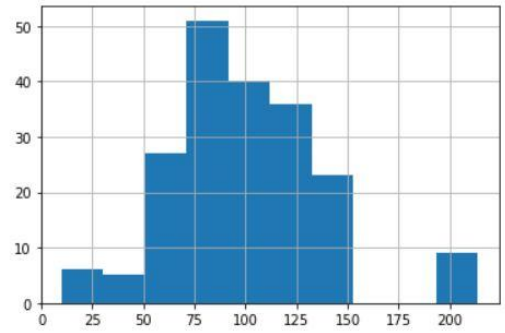
سپس خواص آماری داده‌های آموزشی و آزمایشی را بدست می‌آوریم.

خواص آماری داده‌های آموزشی: (محور ایکس تعداد کلمه در هر متن و محور ایگرگ تعداد متن‌های دارای تعداد کلمات مشخص هستند)




```
count    4190.000000
mean      106.225537
std        49.382846
min         9.000000
25%        77.000000
50%        98.000000
75%       124.000000
max       556.000000
dtype: float64
```

و خواص آماری داده‌های آموزشی: (محور ایکس تعداد کلمه در هر متن و محور ایگرگ تعداد متن‌های دارای تعداد کلمات مشخص هستند)




```
count    197.000000
mean     101.091371
std       38.807765
min       10.000000
25%       77.000000
50%       99.000000
75%      126.000000
max      214.000000
dtype: float64
```

در قسمت بعدی، تمامی متون خیلی کوتاه یا بلند که نوعی Outlier محسوب می‌شوند را حذف می‌کنیم:

```
text_int_train = [ text_int_train[i] for i, l in enumerate(train_title_len) if l>0 ]
encoded_labels_train = [ augmented_train[1][i] for i, l in enumerate(train_title_len) if l> 0 ]
```

سپس تمامی متون را به شکل یک متن با اندازه ثابت 100 کلمه‌ای(هر کلمه یک بردار است) تبدیل می‌کنیم.
توجه شود که متون کوتاه‌تر با بردارهای کاملا صفر پر شده و متون بلندتر از انتها به اندازه لازم بریده می‌شوند.

```
def pad_features(reviews_int, seq_length):
    ''' Return features of review_ints, where each review is padded with 0's or truncated to the input seq_length.
    '''
    features = np.zeros((len(reviews_int), seq_length), dtype = int)
    for i, review in enumerate(reviews_int):
        review_len = len(review)
        if review_len <= seq_length:
            zeroes = list(np.zeros(seq_length-review_len))
            new = np.append(zeroes, review)
        elif review_len > seq_length:
            new = review[0:seq_length]

        features[i,:] = np.array(new)

    return features
```

در مرحله بعد نیز با استفاده از متد get_dummies کتابخانه Pandas، برچسب هارا به شکل یک بردار با اندازه ۳ و فرمت One-hot encoding تبدیل می‌کنیم و هم‌چنین تمامی داده‌هارا تبدیل به فرمت آرایه‌های کتابخانه Numpy در می‌آوریم.

```
'''
convert train and test set to numpy arrays and create one-hot encoding
format of labels
'''

train_x = np.array(text_int_train)
train_y = np.array(encoded_labels_train)
train_y = pd.get_dummies(encoded_labels_train).values

test_x = np.array(text_int_test)
test_y = np.array(encoded_labels_test)
test_y = pd.get_dummies(encoded_labels_test).values
```

فاز دوم: تعریف مدل یادگیری ژرف و آموزش این مدل با داده‌ها:

در این فاز ابتدا مشخص می‌کنیم که هر کلمه به یک بردار با طول ۴۰۰ در فضای تعبیه تبدیل شده و مدل یادگیری ژرف به شکل زیر باشد:

```
model = Sequential()
model.add(Embedding(vocab_size, embedding_dim, input_length=100))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(embedding_dim, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(3, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 400)          6818000

 spatial_dropout1d (SpatialD  (None, 100, 400)         0
 ropout1D)

 lstm (LSTM)                 (None, 400)               1281600

 dense (Dense)               (None, 3)                 1203

=================================================================
Total params: 8,100,803
Trainable params: 8,100,803
Non-trainable params: 0
```

مدل را در ۵ دوره و با Batch_size برابر با ۶۴ آموزش می‌دهیم. دقت و خطاهای گزارش شده به شکل زیر است:

```
Epoch 1/5
59/59 [==============================] - 185s 3s/step - loss: 0.9998 - accuracy: 0.5243 - val_loss: 1.0345 - val_accuracy: 0.4964
Epoch 2/5
59/59 [==============================] - 168s 3s/step - loss: 0.7167 - accuracy: 0.7003 - val_loss: 1.0607 - val_accuracy: 0.5322
Epoch 3/5
59/59 [==============================] - 168s 3s/step - loss: 0.5555 - accuracy: 0.7552 - val_loss: 1.1059 - val_accuracy: 0.5251
Epoch 4/5
59/59 [==============================] - 167s 3s/step - loss: 0.5014 - accuracy: 0.7616 - val_loss: 1.1267 - val_accuracy: 0.5203
```

مشاهده می‌شود که دقت آموزشی 76 درصد و دقت اعتبارسنجی 52 درصد حاصل می‌شود.

فاز سوم: آزمایش مدل آموزش دیده‌شده برروی دادگان آزمایشی

در آخر این مدل را روی داده‌های آزمایشی اعمال کرده و مشاهده می‌شود که دقت آزمایشی و نهایی 55 درصد بدست آورده می‌شود.

```
7/7 [==============================] - 3s 384ms/step - loss: 1.0466 - accuracy: 0.5533
Test set
  Loss: 1.047
  Accuracy: 0.553
```