به نام خدا

صادق جعفری – امین متوسلی

مدل ساده:

ابتدا داده‌ها از فایل data.json استخراج می‌شود:

with open("data.json") as f:

  data = json.load(f)

codes = []

comments = []

for sample in data:

  codes.append(sample["method\_text"])

  comments.append(sample["comment\_text"])

سپس یک سری اطلاعات آماری از روی داده‌های به دست می‌آید تا یک دید نسبی نسبت به داده پیدا شود:

method\_tokens\_counter = collections.Counter([token for method in methods for token in method])

comment\_words\_counter = collections.Counter([word for comment in comments for word in comment])

print('{} Method words.'.format(len([token for method in methods for token in method])))

print('{} unique Method words.'.format(len(method\_tokens\_counter)))

print('10 Most common words in the Method dataset:')

print('"' + '" "'.join(list(zip(\*method\_tokens\_counter.most\_common(10)))[0]) + '"')

print()

print('{} Comment words.'.format(len([word for comment in comments for word in comment])))

print('{} unique Comment words.'.format(len(comment\_words\_counter)))

print('10 Most common words in the Comment dataset:')

print('"' + '" "'.join(list(zip(\*comment\_words\_counter.most\_common(10)))[0]) + '"')

--------------------------------------------------------------

446954 Method words.

12020 unique Method words.

10 Most common words in the Method dataset:

"(" ")" ";" "." "{" "}" "," "=" "if" "return"

240016 Comment words.

6908 unique Comment words.

10 Most common words in the Comment dataset:

"\*" "the" "." "@" ">" "<" "of" "," "param" "to"

در مرحله بعد vocabulary ساخته شده و به هر کلمه در داخل آن یک index اختصاص داده می‌شود:

comment\_words\_index = {}

counter = 1

for word in comment\_words\_counter:

  comment\_words\_index[word] = counter

  counter += 1

method\_tokens\_index = {}

counter = 1

for token in method\_tokens\_counter:

  method\_tokens\_index[token] = counter

  counter += 1

در این مرحله یک تعداد تابع برای پیش پردازش داده اعم از "تبدیل index به token", "تبدیل token به index" و "pad کردن یک جمله" تعریف شده است:

def convert\_tokens\_to\_index(data, index\_dic):

  result = []

  for sample in data:

    result.append(np.array([index\_dic[key] for key in sample]))

  return result

def convert\_index\_to\_tokens(data, token\_list):

  result = []

  for sample in data:

    result.append(" ".join([token\_list[index - 1] for index in sample]))

  return result

def pad(x, length=None):

    if length is None:

        length = max([len(sentence) for sentence in x])

    return pad\_sequences(x, maxlen = length, padding = 'post')

در این مرحله tokenها تبدیل به index می‌شوند:

methods\_index = convert\_tokens\_to\_index(methods, method\_tokens\_index)

comments\_index = convert\_tokens\_to\_index(comments, comment\_words\_index)

در این مرحله یک مدل encoder-decoder برای ترجمه ماشینی تعریف شده است:

def model\_final(input\_shape, output\_sequence\_length, methods\_vocab\_size, comments\_vocab\_size):

  model = Sequential()

  model.add(Embedding(input\_dim=methods\_vocab\_size,output\_dim=128,input\_length=input\_shape[1]))

  model.add(Bidirectional(GRU(256,return\_sequences=False)))

  model.add(RepeatVector(output\_sequence\_length))

  model.add(Bidirectional(GRU(256,return\_sequences=True)))

  model.add(TimeDistributed(Dense(comments\_vocab\_size,activation='softmax')))

  learning\_rate = 0.005

  model.compile(loss = sparse\_categorical\_crossentropy,

                optimizer = Adam(learning\_rate),

                metrics = ['accuracy'])

  return model

در این مرحله آموزش مدل برای 10 epoch انجام می‌شود:

tmp\_X = pad(methods\_index)

tmp\_Y = pad(comments\_index)

model = model\_final(tmp\_X.shape,

                    tmp\_Y.shape[1],

                    len(comment\_words\_counter)+1,

                    len(comment\_words\_counter)+1)

model.fit(tmp\_X, tmp\_Y, batch\_size = 64, epochs = 10, validation\_split = 0.2)

در مرحله آخر کامنت یک متد پیشبینی می‌شود:

def final\_predictions(x\_shape):

  y\_id\_to\_word = {value: key for key, value in comment\_words\_index.items()}

  y\_id\_to\_word[0] = '<PAD>'

  sentence = methods\_index[5010]

  sentence = pad\_sequences([sentence], maxlen=x\_shape, padding='post')

  predictions = model.predict(sentence, 1)

  print('Sample 1:')

  for p in predictions:

    print(' '.join([y\_id\_to\_word[np.argmax(i)] for i in p]))

final\_predictions(tmp\_X.shape[-1])

مدل پیچیده:

در این روش با استفاده از fine tune کردن یک transformer از پیش آموزش دیده به نتایج بهتری دست پیدا می‌شود:

ابتدا tokenizer, model مشخص شده است:

model\_checkpoint = "SEBIS/code\_trans\_t5\_small\_code\_comment\_generation\_java\_transfer\_learning\_finetune"

tokenizer = AutoTokenizer.from\_pretrained(model\_checkpoint)

model = AutoModelForSeq2SeqLM.from\_pretrained(model\_checkpoint)

سپس متریک bleu به عنوان معیار ارزیابی مشخص شده است:

bleu\_metric = load\_metric("bleu")

در مرحله بعد داده‌ها از فایل data.json استخراج می‌شود:

with open("data.json") as f:

  data = json.load(f)

codes = []

comments = []

for sample in data:

  codes.append(sample["method\_text"])

  comments.append(sample["comment\_text"])

در این مرحله preprocessهای لازم روی داده اعمال می‌شود:

max\_input\_length = 512

max\_target\_length = 512

source\_input = "code"

target\_output = "comment"

def preprocess\_function(examples):

    inputs = examples[source\_input]

    targets = examples[target\_output]

    model\_inputs = tokenizer(inputs, max\_length=max\_input\_length, truncation=True)

    # Setup the tokenizer for targets

    with tokenizer.as\_target\_tokenizer():

        labels = tokenizer(targets, max\_length=max\_target\_length, truncation=True)

    model\_inputs["labels"] = labels["input\_ids"]

    return model\_inputs

در مرحله بعد با استفاده از tokenizer متن‌ها را tokenize کرده و indexing نیز انجام داده شده است:

def get\_tokenized\_datasets(codes, comments, train, val, test):

  no\_data = len(codes)

  train\_data = []

  for i in range(0, int(no\_data\*train)):

    train\_data.append(preprocess\_function({"code":codes[i], "comment":comments[i]}))

  #train\_data = preprocess\_function(train\_data)

  val\_data = []

  for i in range(int(no\_data\*train), int(no\_data\*(train + val))):

    val\_data.append(preprocess\_function({"code":codes[i], "comment":comments[i]}))

  #val\_data = preprocess\_function(val\_data)

  test\_data = []

  for i in range(int(no\_data\*(train + val)), int(no\_data\*(train + val + test))):

    test\_data.append(preprocess\_function({"code":codes[i], "comment":comments[i]}))

  #test\_data = preprocess\_function(test\_data)

  return {"train":train\_data, "validation":val\_data, "test":test\_data}

در مرحله بعد آرگمان‌های مورد نیاز برای آموزش مدل تعریف شده است:

batch\_size = 8

model\_name = model\_checkpoint.split("/")[-1]

args = Seq2SeqTrainingArguments(

    f"{model\_name}-finetuned-{source\_input}-to-{target\_output}",

    evaluation\_strategy = "epoch",

    learning\_rate=1e-4,

    warmup\_ratio=0.1,

    per\_device\_train\_batch\_size=batch\_size,

    per\_device\_eval\_batch\_size=batch\_size,

    weight\_decay=0.01,

    save\_total\_limit=3,

    num\_train\_epochs=10,

    predict\_with\_generate=True,

    fp16=False,

    fp16\_opt\_level="02",

    push\_to\_hub=False,

    gradient\_accumulation\_steps=32,

    seed=42,

    load\_best\_model\_at\_end=True,

    metric\_for\_best\_model="eval\_bleu",

    greater\_is\_better=True,

    save\_strategy="epoch"

)

در این مرحله یک تابع برای محاسبه متریک مورد نظر نوشته شده است که داده‌ها را گرفته و با توجه به امتیاز bleu را محاسبه می‌کند:

def postprocess\_text(preds, labels):

    preds = [pred.strip().split() for pred in preds]

    labels = [[label.strip().split()] for label in labels]

    return preds, labels

def compute\_metrics(eval\_preds):

    preds, labels = eval\_preds

    if isinstance(preds, tuple):

        preds = preds[0]

    decoded\_preds = tokenizer.batch\_decode(preds, skip\_special\_tokens=True)

    # Replace -100 in the labels as we can't decode them.

    labels = np.where(labels != -100, labels, tokenizer.pad\_token\_id)

    decoded\_labels = tokenizer.batch\_decode(labels, skip\_special\_tokens=True)

    # Some simple post-processing

    decoded\_preds, decoded\_labels = postprocess\_text(decoded\_preds, decoded\_labels)

    result = bleu\_metric.compute(predictions=decoded\_preds, references=decoded\_labels)

    result = {"bleu": result["bleu"]\*100}

    prediction\_lens = [np.count\_nonzero(pred != tokenizer.pad\_token\_id) for pred in preds]

    result["gen\_len"] = np.mean(prediction\_lens)

    result = {k: round(v, 4) for k, v in result.items()}

    return result

در مرحله بعد یک trainer seq2seq تعریف شده است:

trainer = Seq2SeqTrainer(

    model,

    args,

    train\_dataset=tokenized\_datasets["train"],

    eval\_dataset=tokenized\_datasets["validation"],

    data\_collator=data\_collator,

    tokenizer=tokenizer,

    compute\_metrics=compute\_metrics

)

در این مرحله مدل آموزش داده می‌شود:

trainer.train()

در این مرحله یک pipeline برای مدل اصلی و یک pipeline برای مدل fine tune شده تعریف می‌شود:

original\_pipeline = SummarizationPipeline(

    model=model,

    tokenizer=tokenizer,

    device=0

)

pipeline = SummarizationPipeline(

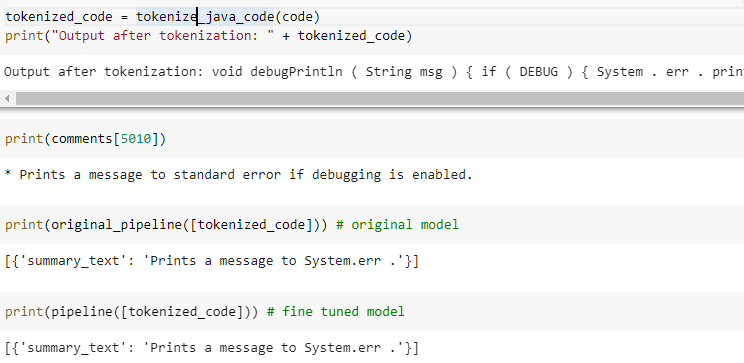
    model=trainer.model,

    tokenizer=tokenizer,

    device=0

)

در مرحله آخر نتایج قبل و بعد از fine tune کردن مدل مشاهده می‌شود:



لینک github:

https://github.com/sadeghjafari5528/CommentGenerationForCode