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 epoch 10 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_fine
tune"
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 seq2seq trainer 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:

```
tokenized_code = tokenize_java_code(code)
print("Output after tokenization: " + tokenized_code)
```

```
Output after tokenization: void debugPrintln ( String msg ) { if ( DEBUG ) { System . err . prin
```

```
print(comments[5010])
```

```
* Prints a message to standard error if debugging is enabled.
```

```
print(original_pipeline([tokenized_code])) # original model
```

```
[{'summary_text': 'Prints a message to System.err .'}]
```

```
print(pipeline([tokenized_code])) # fine tuned model
```

```
[{'summary_text': 'Prints a message to System.err .'}]
```

لینک github:

https://github.com/sadeghjafari5528/CommentGenerationForCode