Chapter 4

다중 개체명 인식

이상민

목차

1. 다중언어 개체명 인식

– 다중언어개체명인식: 여러 언어로 구성된 말뭉치로부터 아래 예시와 같이 **사람, 조직, 위치**와 같은 개체명을 식별하는 **NLP작업**으로 다양한 어플리케이션에 사용된다.

Tokens	2.000	Einwohnern	an	der	Danziger	Bucht	in	der	polnischen	Woiwodschaft	Pommern	.
Tags	O	O	O	O	B-LOC	I-LOC	O	O	B-LOC	B-LOC	I-LOC	O

1. 다중언어 개체명 인식

-XLM-RoBERTa:이번 챕터에서 사용할 **XLM - RoBERTa** 모델은 **100개의 언어**로 구성된 2.5TB 대규모 데이터로 사전학습된 **다중언어 모델**이다. 다중언어로 학습된 모델은 **제로샷 교차 언어 전이**가 가능하다.

-zero-shot transfer or zero-shot learning: **한 레이블 집합에서 모델을 훈련**한 뒤 **다른 레이블 집합에서 평가**하는 작업을 의미함.

1. 다중언어 개체명 인식

PAN-X: 모델을 **fine tune**하기 위해 **WikiANN** 또는 **PAN-X**라 불리는 데이터 셋을 사용한다. **PAN-X 데이터**는 스위스에서 사용되는 **독일어(62.9%), 프랑스어(22.9%), 이탈리아어(8.4%), 영어(5.9%)** 네 가지 언어로 작성된 **wikipedia** 문서로 구성되어있다. 각 문서는 **IOB2** 포맷을 따른다

	0	1	2	3	4	5	6	7	8	9
Tokens	Jeff	Dean	is	a	computer	scientist	at	Google	in	California
Tags	B-PER	I-PER	O	O	O	O	O	B-ORG	O	B-LOC

PER(사람), ORG(조직), LOC(위치)의 태그로 나타내며 **B- 접두사**는 개체명의 **시작**, **I- 접두사**는 동일한 개체명에 속해 **연속되는 토큰**, **O 태그**는 토큰이 **어떤 개체에도 속하지 않음**을 나타낸다

2. 데이터셋

```
from collections import defaultdict
from datasets import DatasetDict

langs = ["de","fr","it","en"]
fracs = [0.629,0.229,0.084,0.059]

panx_ch = defaultdict(DatasetDict)# 키가 없으면 DatasetDict을 변환

for lang,frac in zip(langs, fracs):
  ds = load_dataset("xtreme",name=f"PAN-X.{lang}")
  # 각 분할을 언어 비율에 따라 다운 샘플링하고 섞는다
  for split in ds:
    panx_ch[lang][split] = (
        ds[split].shuffle(seed = 0).select(range(int(frac*ds[split].num_rows)))

    )
```

– 데이터 샘플링: 각 **언어**를 load한 뒤 **데이터**에 의도하지 않은 **편향**이 들어가지 않게 **shuffle()**을 사용해서 섞은 뒤 사전에 정의된 비율에 따라 각 언어를 **다운 샘플링**한다..

2. 데이터셋

– **데이터분포확인: 사전 정의된 비율**에 따라 데이터를 **샘플링**하면 **불균형한 데이터 분포를 갖는다.**

.

	de	fr	it	en
Number of training examples	12580	4580	1680	1180

– **분할 별 개체명 빈도**: 각 분할 별 분포가 대체로 동일하기 때문에 일반화능력을 평가하기에 적합할 것으로 보인다.

	LOC	ORG	PER
train	6186	5366	5810
validation	3172	2683	2893
test	3180	2573	3071

3. 토큰화

- 토큰화 파이프라인

토큰화 파이프라인은 일반적으로 **정규화, 사전토큰화, 토크나이저 모델, 사후처리**와 같이 **네 단계**로 진행 된다

원시 문자열:
"Jack Sparrow loves New York!"

→ 정규화 처리:
"jack sparrow loves new york!"

→ 사전토큰:
"jack", "sparrow", "loves", "new", "york", "!"

→ **토크나이저 모델**:
jack, spa, rrow, loves, new, york, !

→ **사후처리**:
[CLS,jack, spa, rrow, loves, new, york, !,SEP]

3. 토큰화

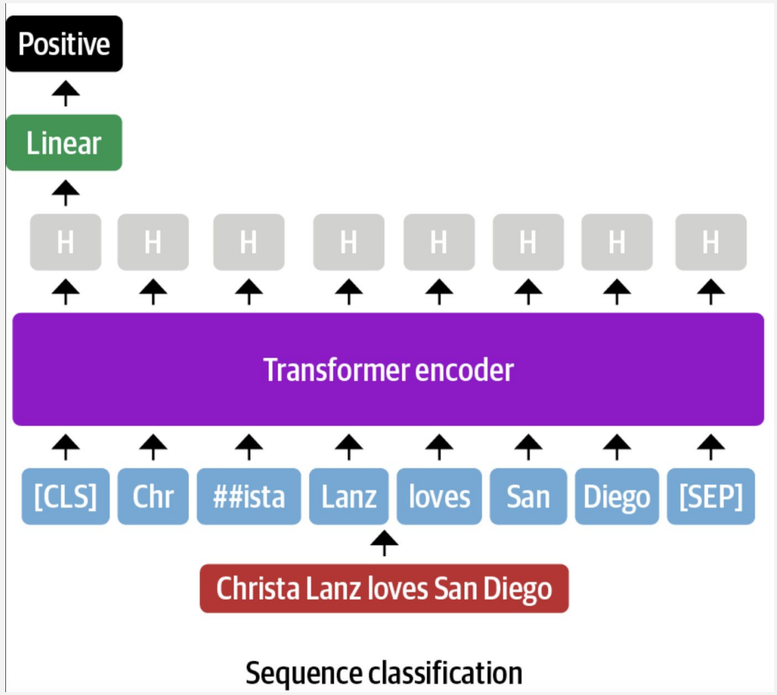
SentencePiece: **SentencePiece** tokenizer는 **사전토큰화** 작업 **없이** 전처리를 하지 않은 **raw data**에 대해 바로 **토큰화**를 수행하므로 **언어**에 **종속** 되지 않는다.

```
words, labels = de_example["tokens"],de_example["ner_tags"]
tokenized_input = xlmr_tokenizer(
    de_example["tokens"],
    is_split_into_words=True#입력 문장이 이미 단어로 나누어 졌다는 사실을 전달.
                                )
tokens = xlmr_tokenizer.convert_ids_to_tokens(tokenized_input["input_ids"])

previous_word_idx = None
label_ids = []

for word_idx in word_ids:
  if word_idx is None or word_idx == previous_word_idx:
    label_ids.append(-100)
  elif word_idx != previous_word_idx:
    label_ids.append(labels[word_idx])
  previous_word_idx = word_idx

labels = [index2tag[l] if l != -100 else "IGN" for l in label_ids]
index = ["Token","Word IDs", "Label IDs", "Labels"]

pd.DataFrame([tokens,word_ids,label_ids,labels],index = index)
```

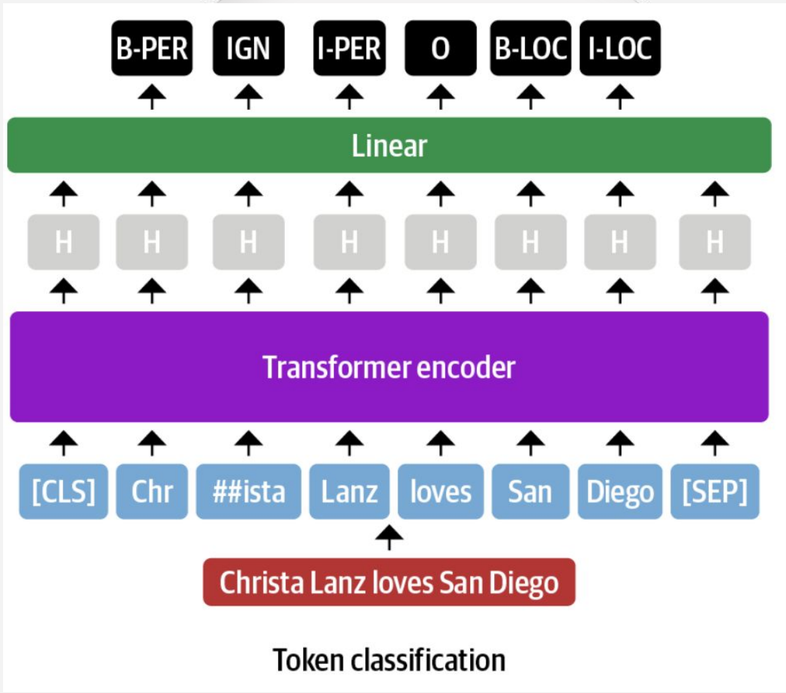
	0	1	2	3	4	5	6	7	8	9	...	15	16	17	18	19	20	21	22	23	24
Token	<s>	__2.000	__Einwohner	n	__an	__der	__Dan	zi	ger	__Buch	...	__Wo	i	wod	schaft	__Po	mmer	n	__	.	</s>
Word IDs	None	0	1	1	2	3	4	4	4	5	...	9	9	9	9	10	10	10	11	11	None
Label IDs	-100	0	0	-100	0	0	5	-100	-100	6	...	5	-100	-100	-100	6	-100	-100	0	-100	-100
Labels	IGN	O	O	IGN	O	O	B-LOC	IGN	IGN	I-LOC	...	B-LOC	IGN	IGN	IGN	I-LOC	IGN	IGN	O	IGN	IGN

4. 개체명 인식을 위한 트랜스포머

<감성분류>



<개체명인식>



감정분류 task에서 **CLS 토큰**을 이용해 **이진 분류**를 했던 것과 달리 NER 작업은 모든 토큰이 Fully Connect layer에 **주입**되어 해당 **토큰**의 **개체명**을 인식한다.

5. 사용자 정의 모델 만들기

```
import torch.nn as nn
from transformers import XLMRobertaConfig
from transformers.modeling_outputs import TokenClassifierOutput
from transformers.models.roberta.modeling_roberta import RobertaModel
from transformers.models.roberta.modeling_roberta import RobertaPreTrainedModel

class XLMRobertaForTokenClassification(RobertaPreTrainedModel):
  config_class = XLMRobertaConfig

  def __init__(self, config):
    super().__init__(config)

    self.num_labels = config.num_labels

    #모델 바디로드
    self.roberta = RobertaModel(
                                config,
                                #[CLS] 토큰에 해당하는 은닉 상태 외 모든 은닉 상태를 반환
                                add_pooling_layer = False
                                )

    #모델 헤드 준비
    self.dropout = nn.Dropout(config.hidden_dropout_prob)
    self.classifier = nn.Linear(config.hidden_size,config.num_labels)
    self.init_weights() #RobertaPreTrainedModel에서 상속된 init_weight메소드 호출

  def forward(self, input_ids = None, attention_mask = None,
```

```
def forward(self, input_ids = None, attention_mask = None,
token_type_ids = None, labels = None, **kwargs):

  #인코도의 출력 결과
  outputs = self.roberta(input_ids, attention_mask = attention_mask,
  token_type_ids = token_type_ids, **kwargs)

  #인코더의 출력결과를 헤드에 입력
  sequence_output = self.dropout(outputs[0])
  logits = self.classifier(sequence_output)
  #loss
  loss = None
  if labels is not None:
    loss_fct = nn.CrossEntropyLoss()
    loss = loss_fct(logits.view(-1,self.num_labels), labels.view(-1))

  return TokenClassifierOutput(loss=loss, logits = logits,
                               hidden_states = outputs.hidden_states,
                               attentions = outputs.attentions)
```

6. XLM-RoBerta 파인튜닝하기

– argument 정의

```
from transformers import TrainingArguments

num_epochs = 3
# 코랩에서 GPU 메모리 부족 에러가 나는 경우 batch_size를 16으로 줄여 주세요.
batch_size = 24  # 16
logging_steps = len(panx_de_encoded["train"]) // batch_size
model_name = f"{xlmr_model_name}-finetuned-panx-de"
training_args = TrainingArguments(
    output_dir=model_name, log_level="error", num_train_epochs=num_epochs,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size, evaluation_strategy="epoch",
    save_steps=1e6, weight_decay=0.01, disable_tqdm=False,
    logging_steps=logging_steps, push_to_hub=True)
```

– data_collator 정의

```
from transformers import DataCollatorForTokenClassification

data_collator = DataCollatorForTokenClassification(xlmr_tokenizer)
```

– model_init 정의

```
def model_init():
    return (XLMRobertaForTokenClassification
            .from_pretrained(xlmr_model_name, config=xlmr_config)
            .to(device))
```

– 인코딩된 정보를 Trainer에 전달

```
from transformers import Trainer

trainer = Trainer(model_init=model_init, args=training_args,
                  data_collator=data_collator, compute_metrics=compute_metrics,
                  train_dataset=panx_de_encoded["train"],
                  eval_dataset=panx_de_encoded["validation"],
                  tokenizer=xlmr_tokenizer)
```

– 학습

```
trainer.train()
```

7. 교차언어 전이

독일어에서 fine tuning된 XLM-R 모델에 대해 다른 언어로 전이 되는 능력을 F1_score기준으로 평가해 보겠습니다.

- 성능평가

```
def evaluate_lang_performance(lang, trainer):
    panx_ds = encode_panx_dataset(panx_ch[lang])
    return get_f1_score(trainer, panx_ds["test"])
```

- f1 score 결과

```
[de] 데이터셋에서 [de] 모델의 F1-점수: 0.867
[fr] 데이터셋에서 [de] 모델의 F1-점수: 0.699
[it] 데이터셋에서 [de] 모델의 F1-점수: 0.649
[en] 데이터셋에서 [de] 모델의 F1-점수: 0.592
```

```
f1_scores = defaultdict(dict)
f1_scores["de"]["de"] = get_f1_score(trainer, panx_de_encoded["test"])
f1_scores["de"]["fr"] = evaluate_lang_performance("fr", trainer)
f1_scores["de"]["it"] = evaluate_lang_performance("it", trainer)
f1_scores["de"]["en"] = evaluate_lang_performance("en", trainer)

print(f"[de] 데이터셋에서 [de] 모델의 F1-점수: {f1_scores['de']['de']:.3f}")
print(f"[fr] 데이터셋에서 [de] 모델의 F1-점수: {f1_scores['de']['fr']:.3f}")
print(f"[it] 데이터셋에서 [de] 모델의 F1-점수: {f1_scores['de']['it']:.3f}")
print(f"[en] 데이터셋에서 [de] 모델의 F1-점수: {f1_scores['de']['en']:.3f}")
```

7. 교차언어 전이

다국어에서 동시에 파인튜닝하기

–네 개의 언어로 구성된 말뭉치 만들기

```
corpora = [panx_de_encoded]

# 반복에서 독일어는 제외합니다.
for lang in langs[1:]:
    training_args.output_dir = f"xlm-roberta-base-finetuned-panx-{lang}"
    # 단일 언어 말뭉치에서 미세 튜닝합니다.
    ds_encoded = encode_panx_dataset(panx_ch[lang])
    metrics = train_on_subset(ds_encoded, ds_encoded["train"].num_rows)
    # 딕셔너리에 F1-점수를 모읍니다.
    f1_scores[lang][lang] = metrics["f1_score"][0]
    # 단일 언어 말뭉치를 corpora 리스트에 추가합니다.
    corpora.append(ds_encoded)
```

– 학습

```
training_args.logging_steps = len(corpora_encoded["train"]) // batch_size
training_args.output_dir = "xlm-roberta-base-finetuned-panx-all"

trainer = Trainer(model_init=model_init, args=training_args,
    data_collator=data_collator, compute_metrics=compute_metrics,
    tokenizer=xlmr_tokenizer, train_dataset=corpora_encoded["train"],
    eval_dataset=corpora_encoded["validation"])

trainer.train()
```

7. 교차언어 전이

다국어에서 동시에 파인튜닝하기

- 결과

Fine-tune on	de	fr	it	en
de	0.8677	0.7141	0.6923	0.539
all	0.8682	0.8647	0.0857	0.787