트랜스포머를 활용한 자연어 처리

Chapter 4. 다중 언어 개체명 인식

24.07.17 박현빈

개체명 인식

- Named Entity Recognition(NER)
- 텍스트에서 사람, 조직, 위치 같은 개체명을 인식하는 것

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

- 책에서는 스위스 고객을 위해 NER을 수행한다고 가정
  - 스위스의 국어 : 독일어, 프랑스어, 이탈리아어, 로망슈어
  - 로망슈어 대신 영어로 대체
  - 다중 언어 모델인 XML-RoBERTa 모델 fine-tunning

데이터셋

- PAN-X 데이터셋
  - 여러 언어의 위키피디아 문서로 구성
  - 독일어(62.9%), 프랑스어(22.9%), 이탈리아어(8.4%), 영어(5.9%)

```
from datasets import load_dataset

load_dataset("xtreme", name="PAN-X.de")
```

```
DatasetDict({
    train: Dataset({
        features: ['tokens', 'ner_tags', 'langs'],
        num_rows: 20000
    })
    validation: Dataset({
        features: ['tokens', 'ner_tags', 'langs'],
        num_rows: 10000
    })
    test: Dataset({
        features: ['tokens', 'ner_tags', 'langs'],
        num_rows: 10000
    })
})
```

데이터셋

- PAN-X 데이터셋
  - 여러 언어의 위키피디아 문서로 구성
  - 독일어(62.9%), 프랑스어(22.9%), 이탈리아어(8.4%), 영어(5.9%)

```python
from collections import defaultdict
from datasets import DatasetDict

langs = ["de", "fr", "it", "en"]
fracs = [0.629, 0.229, 0.084, 0.059]
# 키가 없는 경우 DatasetDict를 반환합니다.
panx_ch = defaultdict(DatasetDict)

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

```python
import pandas as pd

pd.DataFrame({lang: [panx_ch[lang]["train"].num_rows] for lang in langs},
             index=["Number of training examples"])
```

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

데이터셋

데이터 샘플 형식

```
element = panx_ch["de"]["train"][0]
for key, value in element.items():
    print(f"{key}: {value}")
```

```
tokens: ['2.000', 'Einwohnern', 'an', 'der', 'Danziger', 'Bucht', 'in', 'der',
'polnischen', 'Woiwodschaft', 'Pommern', '.']
ner_tags: [0, 0, 0, 0, 5, 6, 0, 0, 5, 5, 6, 0]
langs: ['de', 'de', 'de', 'de', 'de', 'de', 'de', 'de', 'de', 'de', 'de', 'de']
```

```
tags = panx_ch["de"]["train"].features["ner_tags"].feature
print(tags)
```

```
ClassLabel(names=['O', 'B-PER', 'I-PER', 'B-ORG', 'I-ORG', 'B-LOC', 'I-LOC'],
id=None)
```

int2str

```
def create_tag_names(batch):
    return {"ner_tags_str": [tags.int2str(idx) for idx in batch["ner_tags"]]}

panx_de = panx_ch["de"].map(create_tag_names)
```

```
de_example = panx_de["train"][0]
pd.DataFrame([de_example["tokens"], de_example["ner_tags_str"]],
['Tokens', 'Tags'])
```

	0	1	2	3	4	5	6	7	8	9	10	11
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

XLM-R

- RoBERTa와 동일한 모델 구조
- 100개 언어의 대규모 데이터로 학습 진행
- 어휘사전의 크기 : 250,000

XLM-R 토큰화

- WordPiece 토크나이저(BERT)
  - 공백을 기준으로 문장 분할
  - 공백이 없는 언어에서는 잘 작동하지 않음
- SentencePiece 토크나이저(XLM-R)
  - 공백을 기준으로 문자를 분할하지 않음
  - 공백문자를 문자 '_'에 할당(유니코드 : U+2581)
  - 공백이 없는 언어에서도 잘 동작

```
text = "Jack Sparrow loves New York!"
bert_tokens = bert_tokenizer(text).tokens()
xlmr_tokens = xlmr_tokenizer(text).tokens()
```

```
df = pd.DataFrame([bert_tokens, xlmr_tokens], index=["BERT", "XLM-R"])
df
```

	0	1	2	3	4	5	6	7	8	9
BERT	[CLS]	Jack	Spa	##rrow	loves	New	York	!	[SEP]	None
XLM-R	<s>	_Jack	_Spar	row	_love	s	_New	_York	!	</s>

토큰화 파이프라인

정규화

- 문자열을 더 ”깨끗하게” 만들기 위해 사용
- 공백과 악센트가 붙은 문자 제거
- 유니코드 정규화
- 소문자로 변환
- ‘jack sparrow loves new York!’

토큰화 파이프라인

사전 토큰화(pretokenization)

- 텍스트를 작은 단위로 분할하기 위한 초기 단계
- 단어, 구둣점, 숫자 등으로 분할
- [“jack”, “sparrow”, “loves”, “new”, “york”, “!”]

토큰화 파이프라인

토크나이저 모델

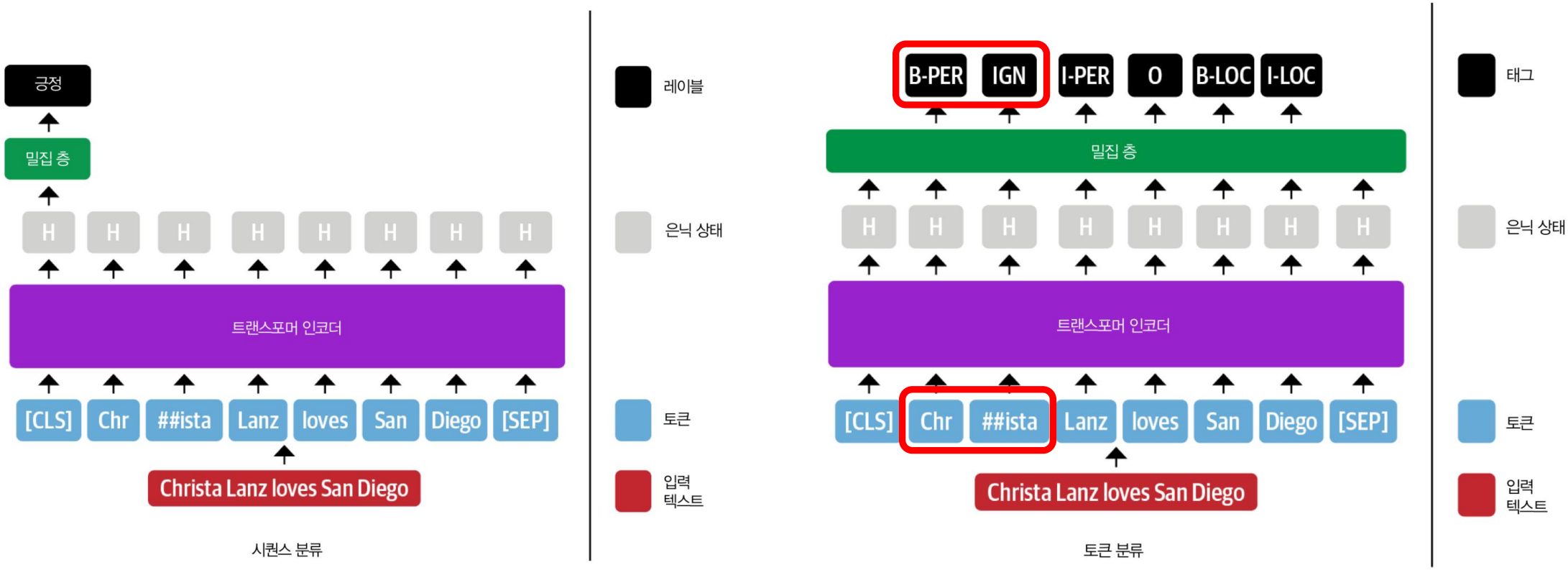
- 훈련 필요
- 단어를 부분단어로 나눠 어휘사전의 크기와 OOV(out-of-vocabulary) 토큰 개수 줄이는 역할
- [jack, spa, rrow, loves, new, york, !]
- 이 시점부터 문자열 리스트가 아닌 정수 리스트

토큰화 파이프라인

사후 처리

- 입력 토큰 인덱스의 처음과 끝에 특수 토큰 추가
- [CLS, jack, spa, rrow, loves, new, york, !, SEP] (BERT)

개체명 인식을 위한 트랜스포머



토큰 분류를 위한 사용자 정의 모델 만들기

```python
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 바디를 로드합니다.
        self.roberta = RobertaModel(config, add_pooling_layer=False)
        # 토큰 분류 헤드를 준비합니다.
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.classifier = nn.Linear(config.hidden_size, config.num_labels)
        # 가중치를 로드하고 초기화합니다.
        self.init_weights()
```

```python
def forward(self, input_ids=None, attention_mask=None, token_type_ids=None,
            labels=None, **kwargs):
    # 모델 바디를 사용해 인코더 표현을 얻습니다.
    outputs = self.roberta(input_ids, attention_mask=attention_mask,
                           token_type_ids=token_type_ids, **kwargs)
    # 인코더 표현을 헤드에 통과시킵니다.
    sequence_output = self.dropout(outputs[0])
    logits = self.classifier(sequence_output)
    # 손실을 계산합니다.
    loss = None
    if labels is not None:
        loss_fct = nn.CrossEntropyLoss()
        loss = loss_fct(logits.view(-1, self.num_labels), labels.view(-1))
    # 모델 출력 객체를 반환합니다.
    return TokenClassifierOutput(loss=loss, logits=logits,
                                 hidden_states=outputs.hidden_states,
                                 attentions=outputs.attentions)
```

사용자 정의 모델 로드

- 필요한 것
  - 태그를 ID로 매핑하는 index2tag 딕셔너리, 그 반대인 tag2index 딕셔너리

```
index2tag = {idx: tag for idx, tag in enumerate(tags.names)}
tag2index = {tag: idx for idx, tag in enumerate(tags.names)}
```

```
from transformers import AutoConfig

xlmr_config = AutoConfig.from_pretrained(xlmr_model_name,
                                         num_labels=tags.num_classes,
                                         id2label=index2tag, label2id=tag2index)
```

```
import torch

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
xlmr_model = (XLMRobertaForTokenClassification
              .from_pretrained(xlmr_model_name, config=xlmr_config)
              .to(device))
```

	0	1	2	3	4	5	6	7	8	9
Tokens	<s>	_Jack	_Spar	row	_love	s	_New	_York	!	</s>
Tags	O	I-LOC	B-LOC	B-LOC	O	I-LOC	O	O	I-LOC	B-LOC

마지막 Linear 모델이 훈련이 안 되었기 때문에 분류가 완벽하지 않음

사용자 정의 모델 로드

```
def tag_text(text, tags, model, tokenizer):
    # 토큰을 준비합니다.
    tokens = tokenizer(text).tokens()
    # 시퀀스를 ID로 인코딩합니다.
    input_ids = xlmr_tokenizer(text, return_tensors="pt").input_ids.to(device)
    # 가능한 일곱 개의 클래스에 대한 분포를 예측합니다.
    outputs = model(input_ids)[0]
    # 토큰마다 가장 가능성 있는 클래스를 argmax로 구합니다.
    predictions = torch.argmax(outputs, dim=2)
    # 데이터프레임으로 변환합니다.
    preds = [tags.names[p] for p in predictions[0].cpu().numpy()]
    return pd.DataFrame([tokens, preds], index=["Tokens", "Tags"])
```

	0	1	2	3	4	5	6	7	8	9
Tokens	<s>	_Jack	_Spar	row	_love	s	_New	_York	!	</s>
Tags	O	I-LOC	B-LOC	B-LOC	O	I-LOC	O	O	I-LOC	B-LOC

NER 작업을 위한 텍스트 토큰화

- 전체 데이터셋 토큰화

```
words
```

```
['2.000',
 'Einwohnern',
 'an',
 'der',
 'Danziger',
 'Bucht',
 'in',
 'der',
 'polnischen',
 'Woiwodschaft',
 'Pommern',
 '.']
```

```
labels
```

```
[0, 0, 0, 0, 5, 6, 0, 0, 5, 5, 6, 0]
```

	0	1	2	3	4	5	6	7	8	9	...	15	16	17	18	19	20	21	22	23	24
Tokens	<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

NER 작업을 위한 텍스트 토큰화

```
words, labels = de_example["tokens"], de_example["ner_tags"]
```

```
tokenized_input = xlmr_tokenizer(de_example["tokens"], is_split_into_words=True)
tokens = xlmr_tokenizer.convert_ids_to_tokens(tokenized_input["input_ids"])
```

```
pd.DataFrame([tokens], index=["Tokens"])
```

	0	1	2	3	4	5	6	7	8	9	...	15	16	17	18	19	20	21	22	23	24
Tokens	<s>	__2.000	__Einwohner	n	__an	__der	__Dan	zi	ger	__Buch	...	__Wo	i	wod	schaft	__Po	mmer	n	__	.	</s>

word_ids()로 하나의 단어였는지 확인

```
word_ids = tokenized_input.word_ids()
pd.DataFrame([tokens, word_ids], index=["Tokens", "Word IDs"])
```

	0	1	2	3	4	5	6	7	8	9	...	15	16	17	18	19	20	21	22	23	24
Tokens	<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

IGN 태그 라벨링

```
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 = ["Tokens", "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
Tokens	<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

지금까지의 과정을 모든 데이터 샘플에 적용

성능 측정

- 정밀도, 재현율, F1-score로 평가 가능
- 토큰 단위가 아닌 개체명 단위로 측정
- Seqeval 라이브러리 사용

```
from seqeval.metrics import classification_report

y_true = [["O", "O", "O", "B-MISC", "I-MISC", "I-MISC", "O"],
          ["B-PER", "I-PER", "O"]]
y_pred = [["O", "O", "B-MISC", "I-MISC", "I-MISC", "I-MISC", "O"],
          ["B-PER", "I-PER", "O"]]
print(classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

        MISC       0.00      0.00      0.00         1
         PER       1.00      1.00      1.00         1

   micro avg       0.50      0.50      0.50         2
   macro avg       0.50      0.50      0.50         2
weighted avg       0.50      0.50      0.50         2
```

XLM-RoBERTa fine-tunning 하기

- Fine-tunning 하기 위해 훈련 진행
  - TrainingArguments 클래스 사용하여 훈련 환경 설정

```
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)
```

XLM-RoBERTa fine-tunning 하기

- 성능 평가 지표로 seqeval 라이브러리의 f1_score 사용

```
from seqeval.metrics import f1_score

def compute_metrics(eval_pred):
    y_pred, y_true = align_predictions(eval_pred.predictions,
                                       eval_pred.label_ids)
    return {"f1": f1_score(y_true, y_pred)}
```

- 배치에서 가장 큰 시퀀스 길이로 입력 시퀀스 패딩
  - DataCollatorForTokenClassification 클래스가 자동으로 수행

```
from transformers import DataCollatorForTokenClassification

data_collator = DataCollatorForTokenClassification(xlmr_tokenizer)
```

XLM-RoBERTa fine-tunning 하기

- Trainer 생성

```
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)
```

- 모델 train

```
trainer.train()
```

Epoch	Training Loss	Validation Loss	F_1
1	0.2652	0.160244	0.822974
2	0.1314	0.137195	0.852747
3	0.0806	0.138774	0.864591

XLM-RoBERTa fine-tunning 하기

- 모델 test

```
text_de = "Jeff Dean ist ein Informatiker bei Google in Kalifornien"
tag_text(text_de, tags, trainer.model, xlmr_tokenizer)
```

	0	1	2	3	4	5	...	8	9	10	11	12	13
Tokens	<s>	_Jeff	_De	an	_ist	_ein	...	_bei	_Google	_in	_Kaliforni	en	</s>
Tags	O	B-PER	I-PER	I-PER	O	O	...	O	B-ORG	O	B-LOC	I-LOC	O

오류 분석

- 슬래시, 괄호, 대문자의 평균 loss가 높다
  - 각 토큰의 Loss 총합이 높은 순으로 나열

	0	1	2	3	4	5	6	7	8	9
input_tokens	_	_der	_in	_von	_/	_und	_(	_)	_"	_A
count	6066	1388	989	808	163	1171	246	246	2898	125
mean	0.03	0.1	0.14	0.14	0.64	0.08	0.3	0.29	0.02	0.44
sum	200.71	138.05	137.33	114.92	104.28	99.15	74.49	72.35	59.31	54.48

- Label로 groupby
  - 모델이 조직 이름의 시작 부분을 결정하기 어려워 함
  - 어디까지가 지역명인지 결정하기 어려워 함

	0	1	2	3	4	5	6
labels	B-ORG	I-LOC	I-ORG	B-LOC	B-PER	I-PER	O
count	2683	1462	3820	3172	2893	4139	43648
mean	0.66	0.64	0.48	0.35	0.26	0.18	0.03
sum	1769.47	930.94	1850.39	1111.03	760.56	750.91	1354.46

오류 분석

- 데이터셋 오류
  - 데이터셋 라벨 자체가 잘못된 경우

	0	1	2	3	4	...	11	12	13	14
tokens	_United	_Nations	_Multi	dimensional	_Integra	...	_Central	_African	_Republic	</s>
labels	B-PER	I-PER	I-PER	IGN	I-PER	...	I-PER	I-PER	I-PER	IGN
preds	B-ORG	I-ORG	I-ORG	I-ORG	I-ORG	...	I-ORG	I-ORG	I-ORG	I-ORG
losses	6.46	5.59	5.51	0.00	5.11	...	5.32	5.10	4.87	0.00

교차 언어 전이

- 독일어로 모델을 fine-tunning 했으므로 다른 언어로 전이되는 능력 평가
  - 독일어 데이터셋에서 독일어 모델의 성능

```
def get_f1_score(trainer, dataset):
    return trainer.predict(dataset).metrics["test_f1"]
```

```
f1_scores = defaultdict(dict)
f1_scores["de"]["de"] = get_f1_score(trainer, panx_de_encoded["test"])
print(f"[de] 데이터셋에서 [de] 모델의 F1-점수: {f1_scores['de']['de']:.3f}")
```

```
[de] 데이터셋에서 [de] 모델의 F1-점수: 0.868
```

  - 프랑스어 데이터셋에서 독일어 모델의 성능
    - 프랑스어 데이터셋 인코딩, 성능 측정

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

```
f1_scores["de"]["fr"] = evaluate_lang_performance("fr", trainer)
print(f"[fr] 데이터셋에서 [de] 모델의 F1-점수: {f1_scores['de']['fr']:.3f}")
```

```
[fr] 데이터셋에서 [de] 모델의 F1-점수: 0.714
```

```
[it] 데이터셋에서 [de] 모델의 F1-점수: 0.692
```

```
[en] 데이터셋에서 [de] 모델의 F1-점수: 0.589
```

다국어로 fine-tunning 하기

- 독일어와 프랑스어 데이터를 합쳐서 훈련
  - concatenate_datasets 클래스 사용

```
[de] 데이터셋에서 [de-fr] 모델의 F1-점수: 0.866
[fr] 데이터셋에서 [de-fr] 모델의 F1-점수: 0.868
[it] 데이터셋에서 [de-fr] 모델의 F1-점수: 0.815
[en] 데이터셋에서 [de-fr] 모델의 F1-점수: 0.677
```

다른 언어의 훈련 데이터를 추가해도, 본 적 없는 언어에서 모델의 성능 향상

Q&A