Ch. 03

트랜스포머 모델을 다루기 위한 허깅페이스 트랜스포머 라이브러리

- 다양한 사전 학습 모델과 데이터셋을 제공하는 온라인 플랫폼

요약

- 트랜스포머의 핵심적인 아키텍처가 공유되었는데도 불구하고 구현 방식에 차이로 인해 활용성에 진입 장벽이 높았으나 허깅페이스의 트랜스포머 라이브러리로 인해 현재 딥러닝 분야의 핵심 라이브러리로 발돋움하게 됨

- 학습 내용
  - 허깅페이스 트랜스포머 라이브러리가 무엇이고 많은 사용자가 허깅페이스를 선호하는 이유
  - 다양한 모델, 데이터셋, 모델 데모를 공유하고 사용 가능한 허깅페이스 허브
  - 모델 학습과 활용에 필요한 핵심 요소: 데이터셋, 모델, 토크나이저
  - 한국어 데이터셋을 활용하여 텍스트 분류 모델을 설계 및 활용 실습

3.1 허깅페이스 트랜스포머란

3.1 허깅페이스 트랜스포머란

- 다양한 트랜스포머 모델을 통일된 인터페이스로 사용할 수 있도록 지원하는 오픈소스 라이브러리
  (BERT, GPT, T5, RoBERTa, DistilBERT 등 다양한 모델을 동일한 방식으로 로드하고 사용 가능)

- 허깅페이스는 크게 두가지 라이브러리를 제공
  - **transformers** 라이브러리 : 트랜스포머 모델과 토크나이저를 활용할 때 사용
  - **datasets** 라이브러리 : 데이터셋을 공개하고 쉽게 가져다 쓸 수 있도록 지원

3.1 허깅페이스 트랜스포머란

```
from transformers import AutoModel, AutoTokenizer

text = "What is Hugging face Transformers?"

# BERT 모델
bert_model = AutoModel.from_pretrained("bert-base-uncased")                # 모델 불러오기
bert_tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')        # 토크나이저 불러오기
encoded_input = bert_tokenizer(text, return_tensors='pt')                  # 입력 토큰화
bert_output = bert_model(**encoded_input)                                  # 모델에 입력
```

✓ 3.0s

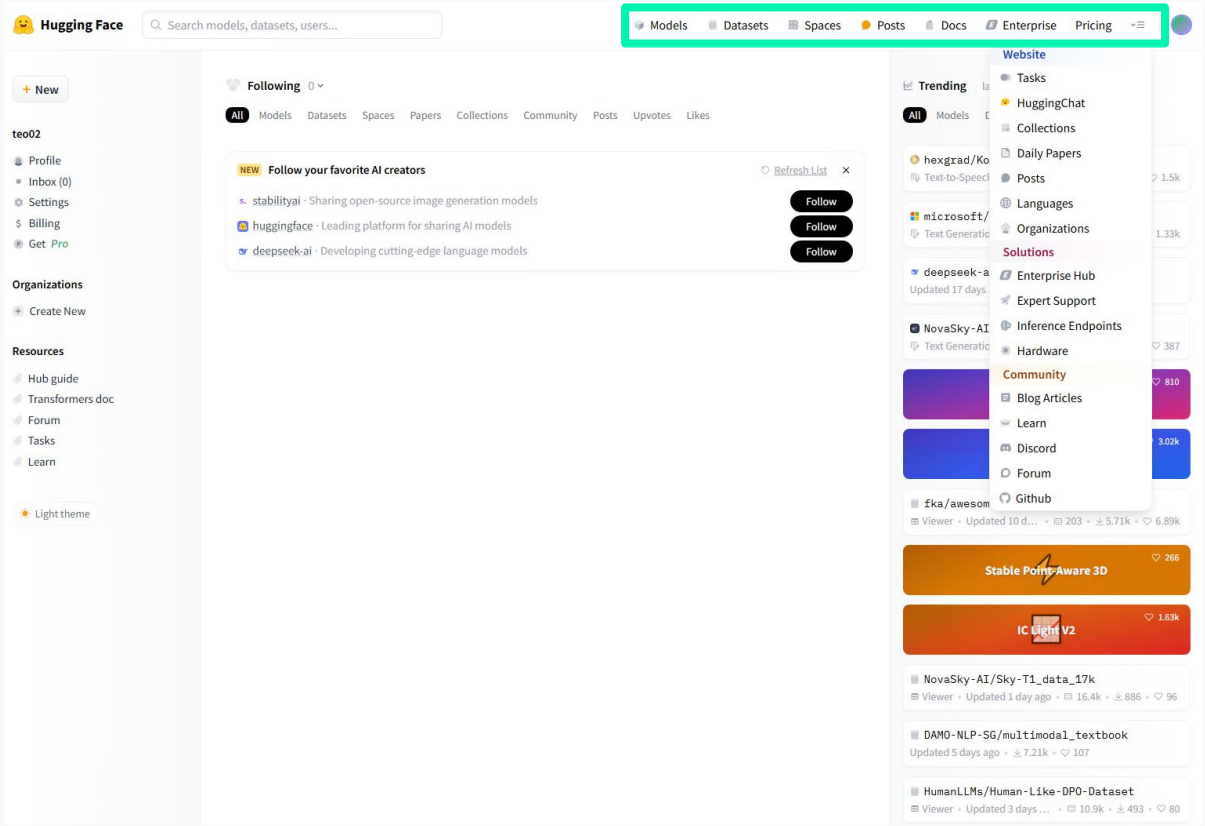
```
# GPT-2 모델
gpt_model = AutoModel.from_pretrained("gpt2")                # 모델 불러오기
gpt_tokenizer = AutoTokenizer.from_pretrained('gpt2')        # 토크나이저 불러오기
encoded_input = gpt_tokenizer(text, return_tensors='pt')     # 입력 토큰화
gpt_output = gpt_model(**encoded_input)                      # 모델에 입력
```

✓ 1.5s

3.2 허깅페이스 허브 탐색하기

- 허깅페이스 허브의 모델, 데이터셋, 스페이스 소개

3.2 허깅페이스 허브



- Model
- Datasets
- Space
- Post: 커뮤니티의 게시물 확인
- Docs: Hugging Face의 다양한 기능과 API 사용법을 설명하는 공식 문서
- Enterprise: 기업 고객을 위한 Hugging Face의 상업적 솔루션을 소개
- Pricing: Hugging Face의 플랜 및 가격 정책
- 기타

3.2.1 모델 허브

- 모델과 데이터셋을 탐색할 수 있는 화면 제공

- 모델 허브에는
  - 어떤 Task에 사용하는 모델인지(컴퓨터 비전, 오디오 처리, 멀티 모달 등)
  - 어떤 언어로 학습된 모델인지

  등 다양한 기준으로 모델이 분류됨

  → 작업 목적에 맞는 사전 학습 모델을 탐색할 수 있고
  그 작업 분야에서 어떤 모델이 많이 사용되는지도 확인 가능

3.2.1 모델 허브

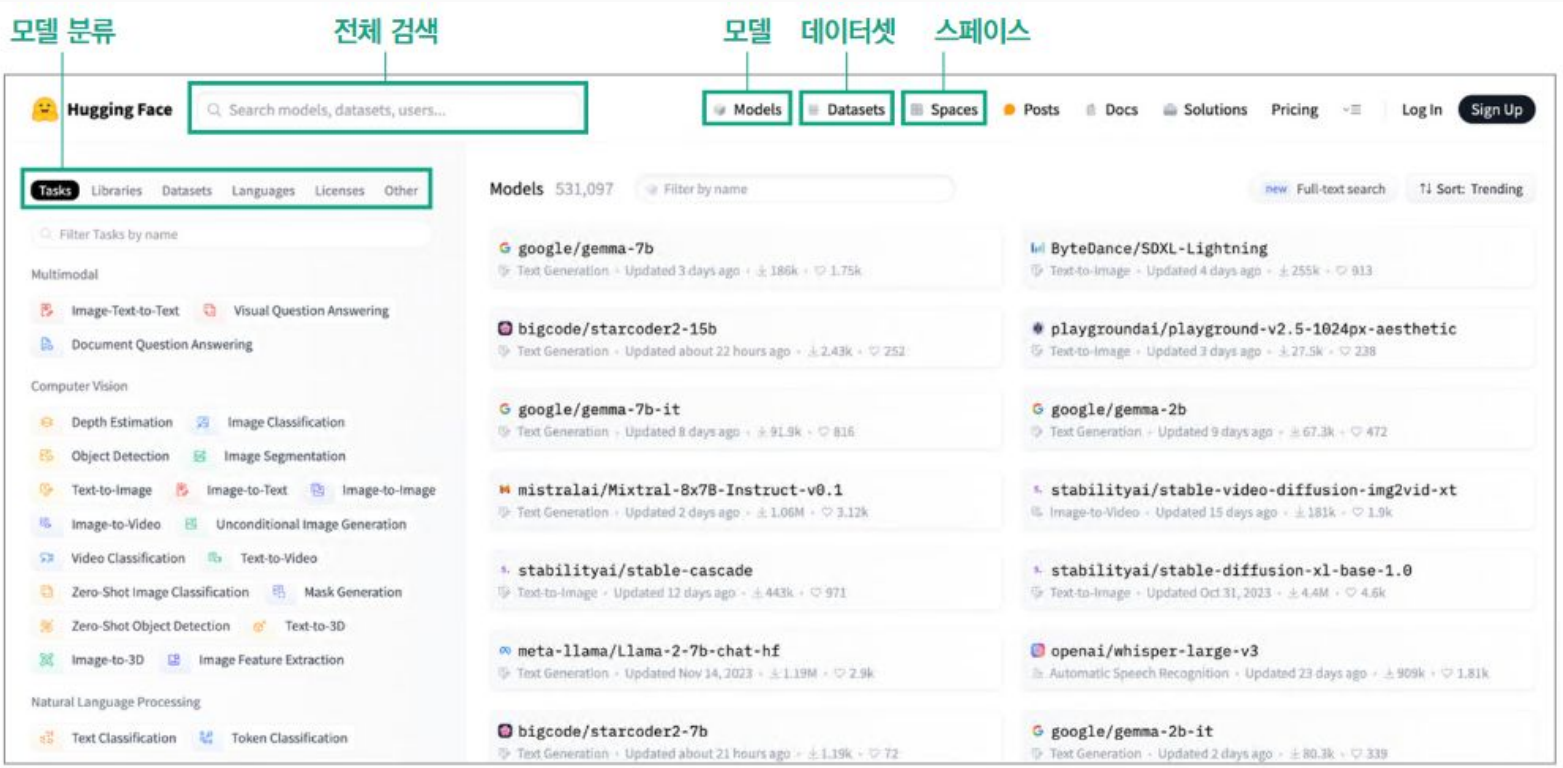


그림 3.1 허깅페이스 모델 허브(https://huggingface.co/models) 스크린샷 화면

3.2.1 모델 허브

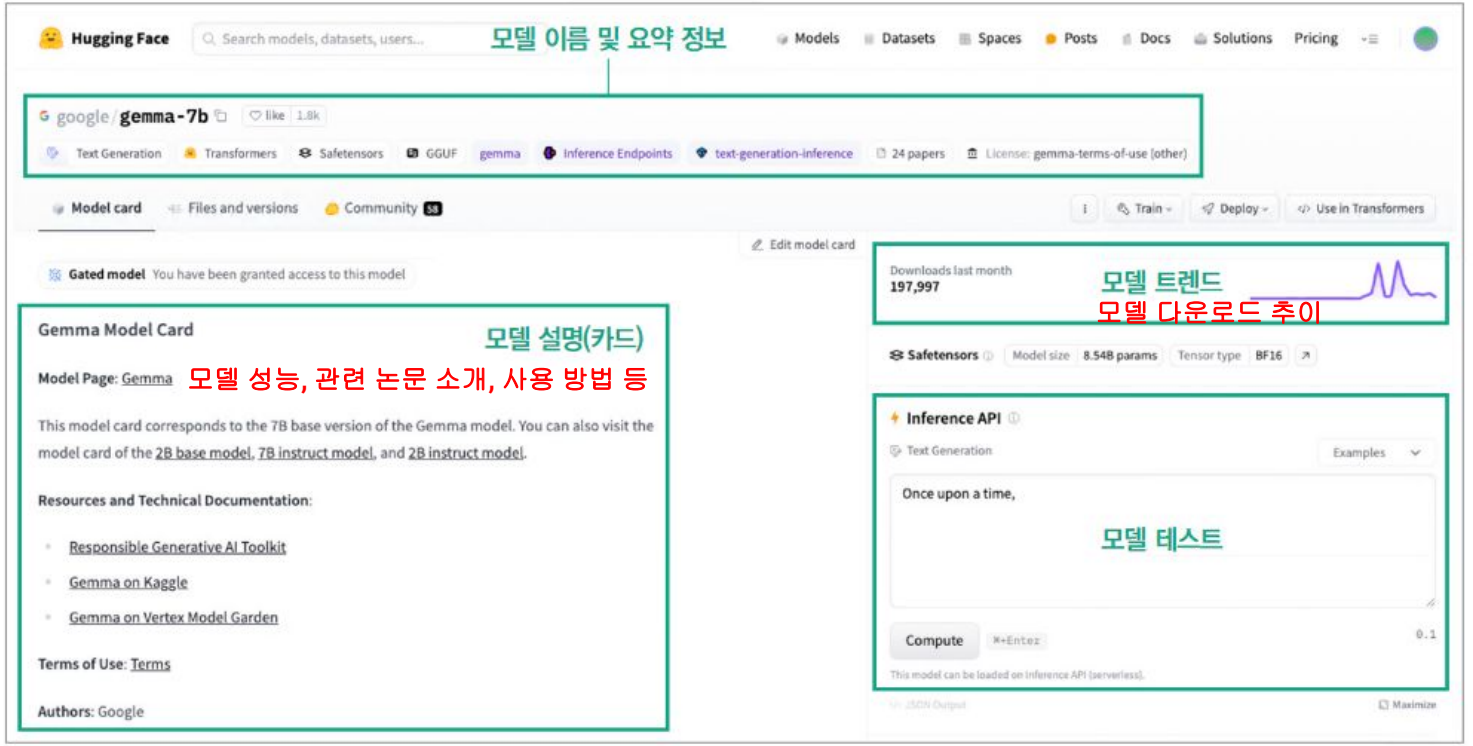


그림 3.2 구글 젬마 모델 페이지 스크린샷 화면(https://huggingface.co/google/gemma-7b) https://huggingface.co/microsoft/phi-4

3.2.2 데이터셋 허브

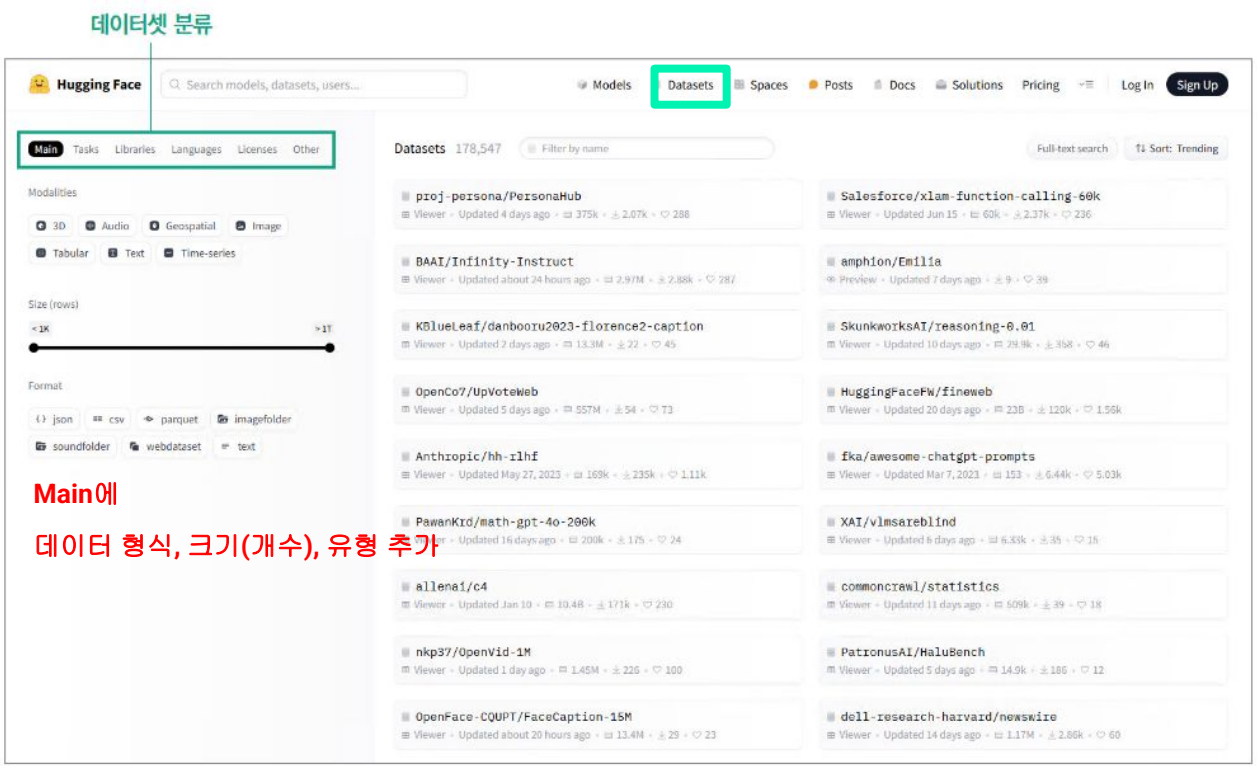


그림 3.3 허깅페이스 데이터셋 허브 스크린샷 화면(https://huggingface.co/datasets)

3.2.2 데이터셋 허브 - KLUE 데이터셋(실습)

Korean Language Understanding Evaluation (KLUE) benchmark: 한국어 NLU를 평가하고 연구하는데 사용되는

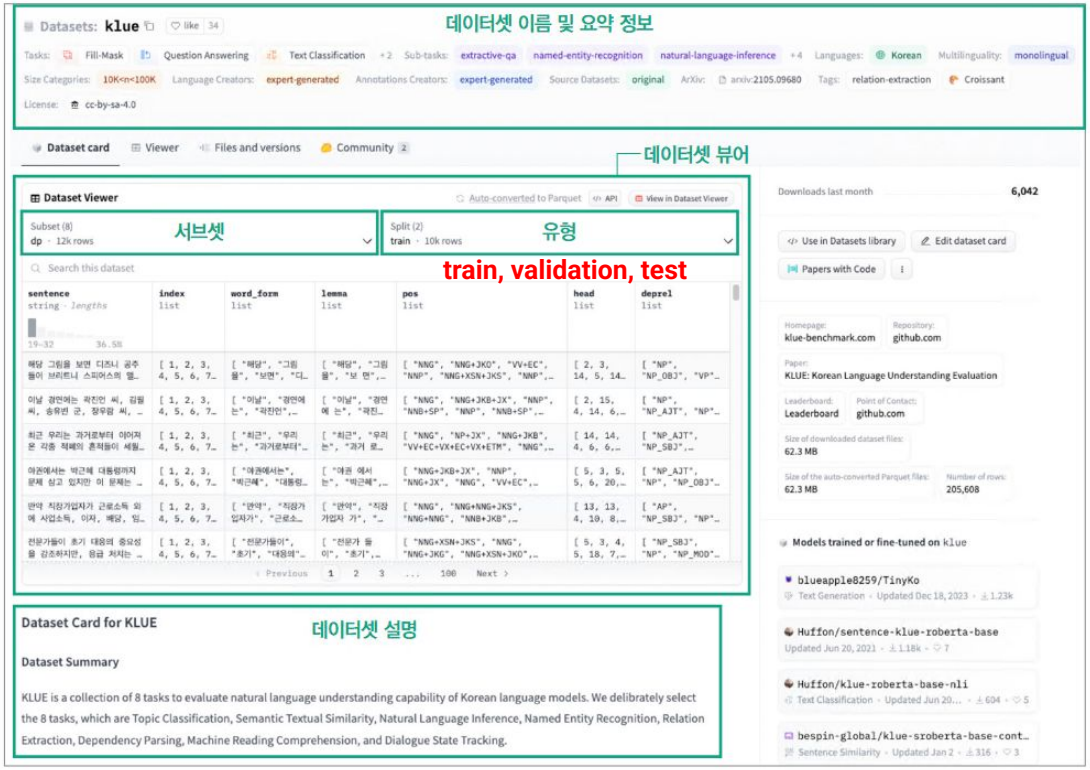


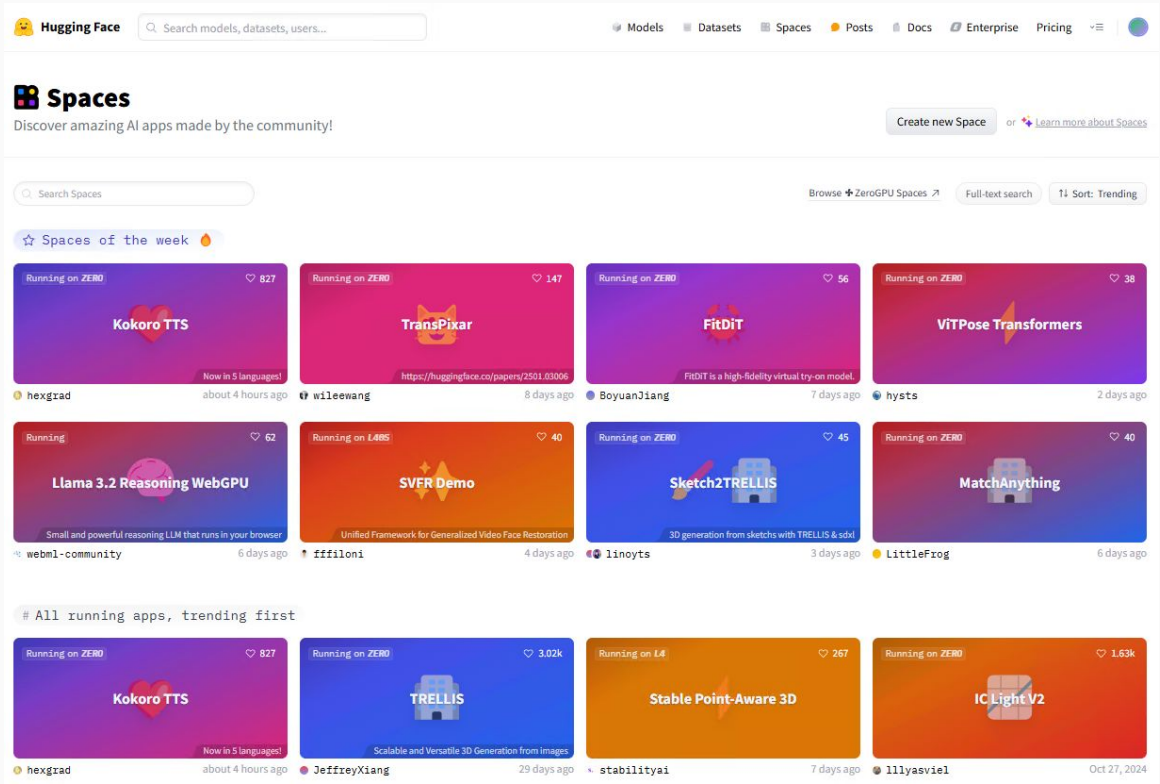
그림 3.4 KLUE 데이터셋 페이지 스크린샷 화면(https://huggingface.co/datasets/klue)

텍스트 분류, 기계 독해, 문장 유사도 판단 등
다양한 작업에서 모델 성능을 평가하기 위해
개발된 벤치마크 데이터셋

작업(Task)	약자	데이터 유형	목적
Topic Classification	Ynat	뉴스 기사 제목	주제 분류
Semantic Textual Similarity	STS	두 문장	문장 간 유사도 측정
Natural Language Inference	NLI	전제와 가설 문장	문장 간 논리적 관계 추론
Named Entity Recognition	NER	문장 내 토큰과 개체명 레이블	개체명 인식
Relation Extraction	RE	문장 내 두 개체와 관계	관계 추출
Dependency Parsing	DP	문장 내 단어와 의존 관계	구문 분석
Machine Reading Comprehension	MRC	지문, 질문, 답변	질문에 대한 답 추출
Dialogue State Tracking	WOS	사용자-시스템 간 대화	대화 상태 추적

3.2.3 모델 데모를 공개하고 사용할 수 있는 스페이스

모델 데모를 공유할 수 있는 웹 페이지



- 개발자가 생성한 웹 애플리케이션 또는 데모를 공유하고 실행
- **Streamlit, Gradio, Flask** 등을 활용하여 제작된 애플리케이션

https://huggingface.co/spaces/whitphx/gradio-lite-transformers-js-yolov9

https://huggingface.co/spaces/fffiloni/SVFR-demo

- 오픈소스 **LLM**과 성능 정보를 게시하는 리더보드 **(**웹페이지 형태이기 때문에…**)**

https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard#/

https://huggingface.co/spaces/upstage/open-ko-llm-leaderboard

3.2.3 모델 데모를 공개하고 사용할 수 있는 스페이스

모델 데모를 공유할 수 있는 웹 페이지

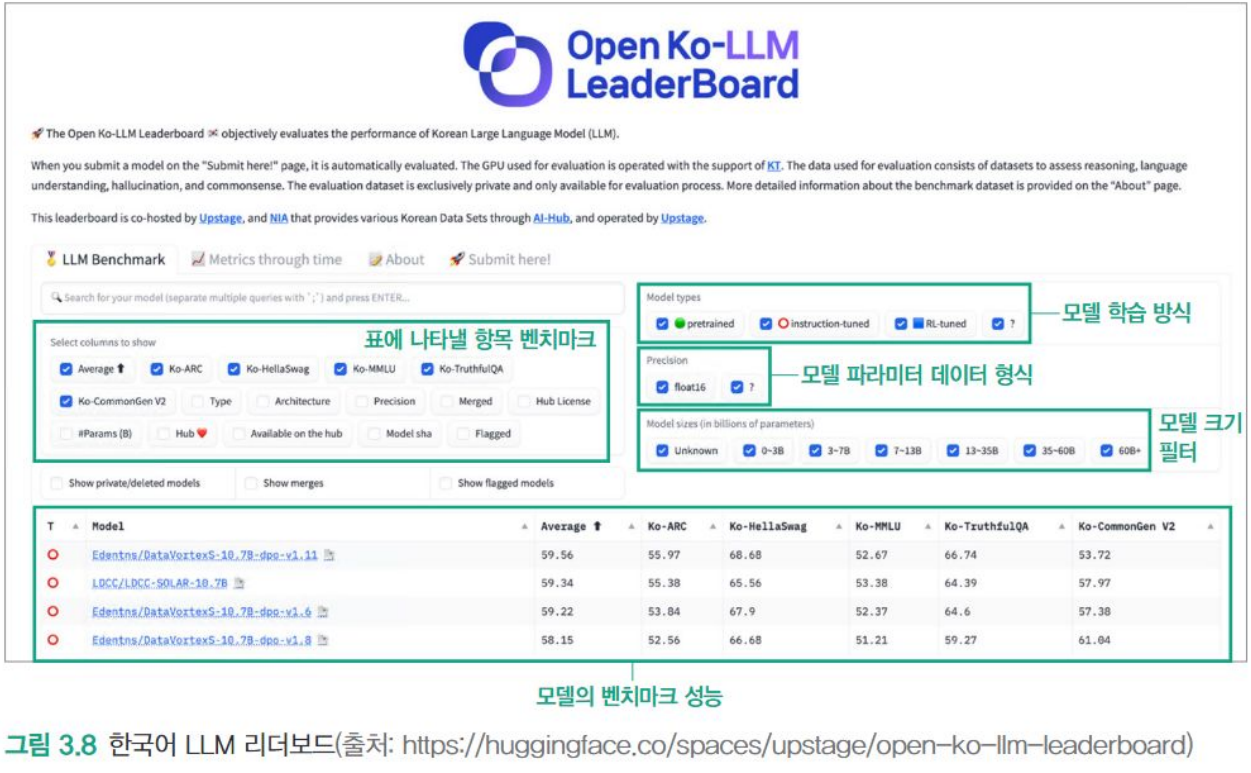


그림 3.8 한국어 LLM 리더보드(출처: https://huggingface.co/spaces/upstage/open-ko-llm-leaderboard)

select columns to show에서

ko-XX는 벤치마크 datasets로 평가한 점수 표시

3.3 허깅페이스 라이브러리 사용법 익히기

3.3.1 모델 활용하기

허깅페이스 모델은 Body와 Head로 구분된다.
왜? 같은 바디를 사용하지만 작업에 따라 서로 다른 헤드를 사용하기 위해...

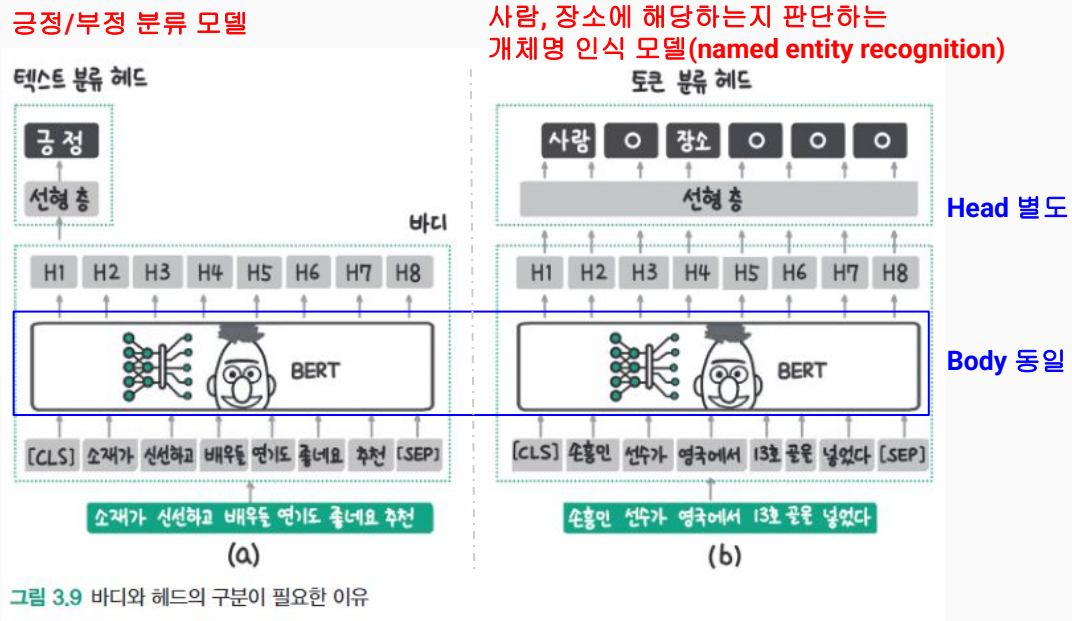


그림 3.9 바디와 헤드의 구분이 필요한 이유

3.3.1 모델 활용하기

Body만 불러오기

```
from transformers import AutoModel
model_id = 'klue/roberta-base'
model = AutoModel.from_pretrained(model_id)
```

✓ 5.8s

```
Some weights of RobertaModel were not initialized from the model checkpoint at klue/roberta-base and are newly initialized: ['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

- **AutoModel:** 모델의 **Body**를 불러오는 클래스
- **model_id**로 클래스를 가져옴
    - 허깅페이스 모델 허브 저장소: **/klue/roberta-base** **RoBERTa 모델을 한국어로 학습**
    - 로컬 경로: **./text_classification**
- **AutoModel**과 **AutoTokenizer**는 **config.json**를 참고해 모델과 토크나이저를 불러옴

파일에 모델 종류, 설정 파라미터, 어휘사전 크기, 토크나이저 클래스 등이 저장

main roberta-base / config.json

james.ryu fix: model_max_length to 514 41b3d9b

raw Copy download link history blame cont


```
{
  "architectures": ["RobertaForMaskedLM"],
  "attention_probs_dropout_prob": 0.1,
  "bos_token_id": 0,
  "eos_token_id": 2,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-05,
  "max_position_embeddings": 514,
  "model_type": "roberta",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 1,
  "type_vocab_size": 1,
  "vocab_size": 32000,
  "tokenizer_class": "BertTokenizer"
}
```

3.3.1 모델 활용하기

텍스트 분류 헤드가 붙은 모델 불러오기

```
from transformers import AutoModelForSequenceClassification
model_id = 'SamLowe/roberta-base-go_emotions'
classification_model = AutoModelForSequenceClassification.from_pretrained(model_id)
```

✓ 3.5s

그림 3.10 텍스트 분류 헤드가 붙은 모델

예제 3.5 텍스트 분류 모델의 config.json 예시

```
{
  "_name_or_path": "roberta-base",
  "architectures": [
    "RobertaForSequenceClassification"
  ],
  ...
  "model_type": "roberta",
  "id2label": {
    "0": "admiration",
    "1": "amusement",
    "2": "anger",
    ...
  }
}
```

https://huggingface.co/SamLowe/roberta-base-go_emotions/blob/main/config.json

3.3.1 모델 활용하기

분류 헤드가 랜덤으로 초기화된 모델 불러오기

```
from transformers import AutoModelForSequenceClassification
model_id = 'klue/roberta-base'
classficationmodel = AutoModelForSequenceClassification.from_pretrained(model_id)
```

✓ 3.7s Python

```
Some weights of RobertaForSequenceClassification were not initialized from the model checkpoint at klue/roberta-base and are newly initialized: ['classifier.dense.bias', 'classifier.dense.weight', 'classifier.out_proj.bias', 'classifi
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

그림 3.11 분류 헤드가 랜덤으로 초기화된 모델

- **Body**는 **klue/roberta-base**의 사전 학습된 파라미터를 불러왔지만 **klue/roberta-base** 모델 허브에

  분류 헤드에 대한 파라미터를 찾을 수 없어 랜덤으로 초기화됨
- 추가 학습 후 사용 필요

실습: 한국어 데이터를 불러와 기사를 적절한 카테고리로 분류하는 모델 개발

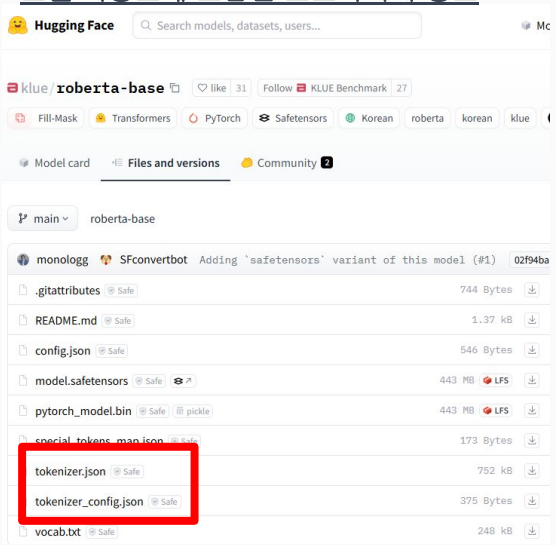
3.3.2 토크나이저 활용하기

- 토크나이저
  - 텍스트를 토큰 단위로 나누고 각 토큰을 대응하는 토큰 아이디로 변환
  - 학습 데이터를 통해 어휘 사전을 구축하기 때문에 일반적으로 모델과 함께 저장

    허깅페이스 허브에서 모델과 토크나이저를 불러오는 경우 동일한 모델 아이디 사용

```
from transformers import AutoTokenizer
model_id = 'klue/roberta-base'
tokenizer = AutoTokenizer.from_pretrained(model_id)
✓ 0.3s
```

모델 저장소에 포함된 트크나이저 정보



3.3.2 토크나이저 활용하기

- 토크나이저 사용 실습

▼ 예제 3.9 토크나이저 사용하기

```
tokenized = tokenizer("토크나이저는 텍스트를 토큰 단위로 나눈다")
print(tokenized) ❶
# {'input_ids': [0, 9157, 7461, 2190, 2259, 8509, 2138, 1793, 2855, 5385, 2200, 20950, 2],
#  'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
#  'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}

print(tokenizer.convert_ids_to_tokens(tokenized['input_ids'])) ❷
# ['[CLS]', '토크', '##나이', '##저', '##는', '텍스트', '##를', '토', '##큰', '단위', '##로',
'나눈다', '[SEP]']

print(tokenizer.decode(tokenized['input_ids'])) ❸
# [CLS] 토크나이저는 텍스트를 토큰 단위로 나눈다 [SEP]

print(tokenizer.decode(tokenized['input_ids'], skip_special_tokens=True)) ❹
# 토크나이저는 텍스트를 토큰 단위로 나눈다
```

토큰 아이디 리스트, 토크나이저 사전의 몇 번째 항목인지
보단내
몇 번째 문장인지 나타냄
실제 텍스트인지 길이를 맞추기 위해 추가한 패딩인지 알려주는 mask

❷ 토큰 아이디를 토큰으로 변환

❸ 토큰 아이디를 텍스트 변환

❹ 토큰 아이디를 텍스트 변환(특수 토큰 제외)

3.3.2 토크나이저 활용하기

- 토크나이저에 여러 문장 넣을 경우

예제 3.10 토크나이지에 여러 문장 넣기

```
tokenizer(['첫 번째 문장', '두 번째 문장'])

# {'input_ids': [[0, 1656, 1141, 3135, 6265, 2], [0, 864, 1141, 3135, 6265, 2]],
# 'token_type_ids': [[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0]],
# 'attention_mask': [[1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1]]}
```

- 한 번에 2개의 문장을 모델에 넣어야 하는 경우(예. 문장이 서로 원인과 결과 관계인지 학습시킬 때)

예제 3.11 하나의 데이터에 여러 문장이 들어가는 경우

```
tokenizer([['첫 번째 문장', '두 번째 문장']])

# {'input_ids': [[0, 1656, 1141, 3135, 6265, 2, 864, 1141, 3135, 6265, 2]],
# 'token_type_ids': [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],
# 'attention_mask': [[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]}
```

- 토큰 아이디를 문자열로 복원

예제 3.12 토큰 아이디를 문자열로 복원

```
first_tokenized_result = tokenizer(['첫 번째 문장', '두 번째 문장'])['input_ids']
tokenizer.batch_decode(first_tokenized_result)
# ['[CLS] 첫 번째 문장 [SEP]', '[CLS] 두 번째 문장 [SEP]']

second_tokenized_result = tokenizer([['첫 번째 문장', '두 번째 문장']])['input_ids']
tokenizer.batch_decode(second_tokenized_result)
# ['[CLS] 첫 번째 문장 [SEP] 두 번째 문장 [SEP]']
```

- **특수토큰 [CLS]으로 문장을 시작하고 [SEP]로 문장을 구분**

※ 특수 토큰은 모델의 아키텍처에 따라 달라질 수 있음

3.3.2 토크나이저 활용하기

- token_type_ids
  - 문장 구분 역할
  - BERT에서 2개 문장이 서로 이어지는지 맞추는 NSP(Nest Sentence Prediction) 작업에 문장을 구분하기 위해 만들어짐

▼ 예제 3.13 BERT 토크나이저와 RoBERTa 토크나이저

```
bert_tokenizer = AutoTokenizer.from_pretrained('klue/bert-base') ❶
bert_tokenizer([['첫 번째 문장', '두 번째 문장']])
# {'input_ids': [[2, 1656, 1141, 3135, 6265, 3, 864, 1141, 3135, 6265, 3]],
# 'token_type_ids': [[0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1]],
# 'attention_mask': [[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]}

roberta_tokenizer = AutoTokenizer.from_pretrained('klue/roberta-base') ❷
roberta_tokenizer([['첫 번째 문장', '두 번째 문장']])
# {'input_ids': [[0, 1656, 1141, 3135, 6265, 2, 864, 1141, 3135, 6265, 2]],
# 'token_type_ids': [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],
# 'attention_mask': [[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]}

en_roberta_tokenizer = AutoTokenizer.from_pretrained('roberta-base') ❸
en_roberta_tokenizer([['first sentence', 'second sentence']])
# {'input_ids': [[0, 9502, 3645, 2, 2, 10815, 3645, 2]],
# 'attention_mask': [[1, 1, 1, 1, 1, 1, 1, 1]]}
```

RoBERTa 계열의 경우 NSP 작업을 학습 과정에서 제거하여 문장 구분이 필요 없음
(더 많은 데이터와 더 긴 문맥을 사용하여 NSP를 제거해도 문장 간 관계를 효과적으로 학습할 수 있음을 증명)
영어 버전의 RoBERTa 경우 token_type_ids가 없음

3.3.2 토크나이저 활용하기

- attention_mask
  - 해당 토큰이 패딩 토큰인지 실제 데이터인지에 대한 정보
  - 패딩: 모델에 입력하는 토큰 아이디의 길이를 맞추기 위해 추가하는 특수 토큰

▼ 예제 3.14 attention_mask 확인

```
tokenizer(['첫 번째 문장은 짧다.', '두 번째 문장은 첫 번째 문장보다 더 길다.'], padding='longest')

# {'input_ids': [[0, 1656, 1141, 3135, 6265, 2073, 1599, 2062, 18, 2, 1, 1, 1, 1, 1, 1],
# [0, 864, 1141, 3135, 6265, 2073, 1656, 1141, 3135, 6265, 3632, 831, 647, 2062, 18, 2]],
# 'attention_mask': [[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
# [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]}
```

longest: 입력 문장 중 가장 긴 문장에 맞춰 패딩 토큰 추가

패딩 특수토큰 확인

```
tokenizer.batch_decode(text2)
✓ 0.0s
['[CLS] 첫 번째 문장은 짧다 [SEP] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]',
 '[CLS] 두 번째 문장은 첫 번째 문장보다 길다 [SEP]']
```

3.3.3 데이터셋 활용하기

KLUE 데이터셋의 서브셋 중 하나인 MRC 데이터셋

▼ 예제 3.15 KLUE MRC 데이터셋 다운로드 https://huggingface.co/datasets/klue/klue

```
from datasets import load_dataset
klue_mrc_dataset = load_dataset('klue', 'mrc')
# klue_mrc_dataset_only_train = load_dataset('klue', 'mrc', split='train')
```

load_dataset 함수로 데이터 MRC 데이터셋 다운로드

train dataset만 받고 싶을 때

MRC 데이터셋 데이터 개수, 컬럼 확인

```
klue_mrc_dataset
✓ 0.0s
DatasetDict({
    train: Dataset({
        features: ['title', 'context', 'news_category', 'source', 'guid', 'is_impossible', 'question_type', 'question', 'answers'],
        num_rows: 17554
    })
    validation: Dataset({
        features: ['title', 'context', 'news_category', 'source', 'guid', 'is_impossible', 'question_type', 'question', 'answers'],
        num_rows: 5841
    })
})
```

3.3.3 데이터셋 활용하기

참고: load_dataset

▼ 예제 3.16 로컬의 데이터 활용하기

```
from datasets import load_dataset
# 로컬의 csv 데이터 파일을 활용
dataset = load_dataset("csv", data_files="my_file.csv") ❶

# 파이썬 딕셔너리 활용
from datasets import Dataset
my_dict = {"a": [1, 2, 3]}
dataset = Dataset.from_dict(my_dict) ❷

# 판다스 데이터프레임 활용
from datasets import Dataset
import pandas as pd
df = pd.DataFrame({"a": [1, 2, 3]})
dataset = Dataset.from_pandas(df) ❸
```

3.4 모델 학습시키기

- 한국어 기사 제목을 바탕으로 기사의 카테고리를 분류 실습
- 모델을 학습시키는 방법
- 허깅페이스 허브 업로드 방법

3.4.1 데이터 준비

- KLUE 데이터셋 YNAT(Yonhap News Article Topic Classification) 서브셋 활용

  한국어 뉴스 기사 제목을 기반으로 주제 분류

```
klue_tc_train = load_dataset('klue', 'ynat', split='train')
klue_tc_eval = load_dataset('klue', 'ynat', split='validation')
klue_tc_train
```

✓ 13.1s

```
Dataset({
    features: ['guid', 'title', 'label', 'url', 'date'],
    num_rows: 45678
})
```

```
klue_tc_train[0]
```

✓ 0.0s

```
{'guid': 'ynat-v1_train_00000',
 'title': '유튜브 내달 2일까지 크리에이터 지원 공간 운영',
 'label': 3,
 'url': 'https://news.naver.com/main/read.nhn?mode=LS2D&mid=shm&sid1=105&sid2=227&oid=001&aid=0008508947',
 'date': '2016.06.30. 오전 10:36'}
```

- guid: 데이터의 고유 ID
- title: 뉴스 제목
- label: 속한 카테고리 ID
- url: 뉴스 링크
- date: 뉴스 입력 시간

```
klue_tc_train.features['label'].names
```

✓ 0.0s

```
['IT과학', '경제', '사회', '생활문화', '세계', '스포츠', '정치']
```

label 컬럼의 항목별 이름

3.4.1 데이터 준비

- 실습에 사용하지 않는 불필요한 컬럼 제거

```
klue_tc_train = klue_tc_train.remove_columns(['guid', 'url', 'date'])
klue_tc_eval = klue_tc_eval.remove_columns(['guid', 'url', 'date'])
klue_tc_train
```

✓ 0.0s

```
Dataset({
    features: ['title', 'label'],
    num_rows: 45678
})
```

3.4.1 데이터 준비

- 분류 카테고리를 확인하기 쉽도록 label_str 컬럼 추가

```
klue_tc_train.features['label']
```

✓ 0.0s

```
ClassLabel(names=['IT과학', '경제', '사회', '생활문화', '세계', '스포츠', '정치'], id=None)
```

```
klue_tc_train.features['label'].int2str(1)
```

✓ 0.0s

```
'경제'
```

```
klue_tc_train.features['label'].str2int('경제')
```

✓ 0.0s

```
1
```

```
klue_tc_label = klue_tc_train.features['label']

def make_str_label(batch):
    batch['label_str'] = klue_tc_label.int2str(batch['label'])
    return batch

klue_tc_train = klue_tc_train.map(make_str_label, batched=True, batch_size=1000)
klue_tc_train[0]
```

✓ 0.0s

```
Map: 100%|██████████| 45678/45678 [00:00<00:00, 1054334.93 examples/s]

{'title': '유튜브 내달 2일까지 크리에이터 지원 공간 운영', 'label': 3, 'label_str': '생활문화'}
```

3.4.1 데이터 준비

- 학습/검증/테스트 데이터셋 분할
  - 학습 데이터 중 10,000개 추출해 사용
  - 테스트 데이터는 검증 데이터셋(klue_tc_eval)에서 1,000개 추출

```
train_dataset = klue_tc_train.train_test_split(test_size=10000, shuffle=True, seed=42)['test']
dataset = klue_tc_eval.train_test_split(test_size=1000, shuffle=True, seed=42)
test_dataset = dataset['test']
valid_dataset = dataset['train'].train_test_split(test_size=1000, shuffle=True, seed=42)['test']
```

✓ 0.0s

3.4.2 트레이너 API를 사용해 학습하기

- 트레이너 API
  - 학습에 필요한 다양한 기능을 학습 인자만으로 쉽게 활용할 수 있도록 허깅페이스에서 제공하는 툴

- Trainer를 사용한 학습: (1) 준비

```python
import torch
import numpy as np
from transformers import (
    Trainer,
    TrainingArguments,
    AutoModelForSequenceClassification,
    AutoTokenizer
)

def tokenize_function(examples):
    return tokenizer(examples["title"], padding="max_length", truncation=True)

model_id = "klue/roberta-base"
model = AutoModelForSequenceClassification.from_pretrained(model_id, num_labels=len(train_dataset.features['label'].names))
tokenizer = AutoTokenizer.from_pretrained(model_id)

train_dataset = train_dataset.map(tokenize_function, batched=True)
valid_dataset = valid_dataset.map(tokenize_function, batched=True)
test_dataset = test_dataset.map(tokenize_function, batched=True)
```

✓ 2.8s Python

```
Some weights of RobertaForSequenceClassification were not initialized from the model checkpoint at klue/roberta-base and are newly initialized:
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Map: 100%|██████████| 10000/10000 [00:01<00:00, 6365.39 examples/s]
Map: 100%|██████████| 1000/1000 [00:00<00:00, 6427.55 examples/s]
Map: 100%|██████████| 1000/1000 [00:00<00:00, 6916.98 examples/s]
```

3.4.2 트레이너 API를 사용해 학습하기

- Trainer를 사용한 학습: (2) 학습 인자와 평가 함수 정의

```
training_args = TrainingArguments(
    output_dir="./results",
    num_train_epochs=1,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    evaluation_strategy="epoch",
    learning_rate=5e-5,
    push_to_hub=False
)

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return {"accuracy": (predictions == labels).mean()}
```

✓ 0.0s

에포크 수
학습 배치 크기
eval 배치 크기
epoch마다 검 데이터셋 으로 평가

학습이 잘 이뤄지고 있는지 확인하기 위한 평가 지표 정의

예측 결과 중 가장 큰 값을 갖는 클래스를 뽑아 predictions에 저장하고 정답이 저장된 label가 같은 값을 갖는 결과의 비율 계산

3.4.2 트레이너 API를 사용해 학습하기

- Trainer를 사용한 학습: (3) 학습 진행

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=valid_dataset,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
)

trainer.train()

trainer.evaluate(test_dataset) # 정확도 0.84
```

```
train_dataset = klue_tc_train.train_test_split(test_size=200, shuffle=True, seed=42)['test']
dataset = klue_tc_eval.train_test_split(test_size=50, shuffle=True, seed=42)
test_dataset = dataset['test']
valid_dataset = dataset['train'].train_test_split(test_size=50, shuffle=True, seed=42)['test']
```

학습 시간 줄이기 위해 데이터 사이즈 변경

✓ 5m 32.5s Python

```
 60%|██████    | 75/125 [21:47<14:31, 17.43s/it]
 34%|███       | 17/50 [09:14<06:29, 11.80s/it]

100%|██████████| 25/25 [05:12<00:00, 12.49s/it]
{'eval_loss': 1.9694219827651978, 'eval_accuracy': 0.24, 'eval_runtime': 21.9066, 'eval_samples_per_second': 2.282, 'eval_steps_per_second': 0.
{'train_runtime': 312.3739, 'train_samples_per_second': 0.64, 'train_steps_per_second': 0.08, 'train_loss': 1.801092071533203, 'epoch': 1.0}
100%|██████████| 7/7 [00:16<00:00,  2.34s/it]

{'eval_loss': 1.9523234367370605,
 'eval_accuracy': 0.28,
 'eval_runtime': 19.659,
 'eval_samples_per_second': 2.543,
 'eval_steps_per_second': 0.356,
 'epoch': 1.0}
```

3.4.3 트레이너 API를 사용하지 않고 학습하기

- 트레이너 API는 간편하지만 내부 동작을 파악하기 어렵다는 단점 존재
- Trainer를 사용하지 않는 학습: (1) 학습을 위한 모델과 토크나이저 준비

```python
import torch
from tqdm.auto import tqdm
from torch.utils.data import DataLoader
from transformers import AdamW

def tokenize_function(examples): # 제목(title) 컬럼에 대한 토큰화
    return tokenizer(examples["title"], padding="max_length", truncation=True)

# 모델과 토크나이저 불러오기
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model_id = "klue/roberta-base"
model = AutoModelForSequenceClassification.from_pretrained(model_id, num_labels=len(train_dataset.features['label'].names))
tokenizer = AutoTokenizer.from_pretrained(model_id)
model.to(device)
```

trainer에서는 GPU로의 모델 이동을 내부적으로 수행

✓ 0.9s Python

```
Some weights of RobertaForSequenceClassification were not initialized from the model checkpoint at klue/roberta-base and are newly initialized:
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
RobertaForSequenceClassification(
  (roberta): RobertaModel(
    (embeddings): RobertaEmbeddings(
      (word_embeddings): Embedding(32000, 768, padding_idx=1)
      (position_embeddings): Embedding(514, 768, padding_idx=1)
      (token_type_embeddings): Embedding(1, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): RobertaEncoder(
      (layer): ModuleList(
        (0-11): 12 x RobertaLayer(
          (attention): RobertaAttention(
            (self): RobertaSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
```

3.4.3 트레이너 API를 사용하지 않고 학습하기

- Trainer를 사용하지 않는 학습: (2) 학습을 위한 데이터 준비

```
def make_dataloader(dataset, batch_size, shuffle=True):
    dataset = dataset.map(tokenize_function, batched=True).with_format("torch") # 데이터셋에 토큰화 수행
    dataset = dataset.rename_column("label", "labels") # 컬럼 이름 변경
    dataset = dataset.remove_columns(column_names=['title']) # 불필요한 컬럼 제거
    return DataLoader(dataset, batch_size=batch_size, shuffle=shuffle)

# 데이터로더 만들기
train_dataloader = make_dataloader(train_dataset, batch_size=8, shuffle=True)
valid_dataloader = make_dataloader(valid_dataset, batch_size=8, shuffle=False)
test_dataloader = make_dataloader(test_dataset, batch_size=8, shuffle=False)
```

파이토치 DataLoader 클래스로 배치 데이터 생성

✓ 0.1s

```
Map: 100%|██████████| 200/200 [00:00<00:00, 2387.85 examples/s]
Map: 100%|██████████| 50/50 [00:00<00:00, 3165.46 examples/s]
Map: 100%|██████████| 50/50 [00:00<00:00, 3424.59 examples/s]
```

3.4.3 트레이너 API를 사용하지 않고 학습하기

- Trainer를 사용하지 않는 학습: (3) 학습을 위한 함수 정의

```
def train_epoch(model, data_loader, optimizer):
    model.train()
    total_loss = 0
    for batch in tqdm(data_loader):
        optimizer.zero_grad()
        input_ids = batch['input_ids'].to(device) # 모델에 입력할 토큰 아이디
        attention_mask = batch['attention_mask'].to(device) # 모델에 입력할 어텐션 마스크
        labels = batch['labels'].to(device) # 모델에 입력할 레이블
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels) # 모델 계산
        loss = outputs.loss # 손실
        loss.backward() # 역전파
        optimizer.step() # 모델 업데이트
        total_loss += loss.item()
    avg_loss = total_loss / len(data_loader)
    return avg_loss
```

학습과 평가에 사용할 함수 생성
모델 학습 모드
loss 계산 및 모델 업데이트

✓ 0.0s

3.4.3 트레이너 API를 사용하지 않고 학습하기

- Trainer를 사용하지 않는 학습: (4) 평가를 위한 함수 정의

```python
def evaluate(model, data_loader):
    model.eval()
    total_loss = 0
    predictions = []
    true_labels = []
    with torch.no_grad():
        for batch in tqdm(data_loader):
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)
            outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
            logits = outputs.logits
            loss = outputs.loss
            total_loss += loss.item()
            preds = torch.argmax(logits, dim=-1)
            predictions.extend(preds.cpu().numpy())
            true_labels.extend(labels.cpu().numpy())
    avg_loss = total_loss / len(data_loader)
    accuracy = np.mean(np.array(predictions) == np.array(true_labels))
    return avg_loss, accuracy
```

추론 모드

가장 큰 값으로 예측한 카테고리 정보를 찾고 실제 정답과 비교

✓ 0.0s

3.4.3 트레이너 API를 사용하지 않고 학습하기

- Trainer를 사용하지 않는 학습: (5) 학습 수행

학습 시간 줄이기 위해 데이터 사이즈 변경하고 학습

```
num_epochs = 1
optimizer = AdamW(model.parameters(), lr=5e-5)

# 학습 루프
for epoch in range(num_epochs):
    print(f"Epoch {epoch+1}/{num_epochs}")
    train_loss = train_epoch(model, train_dataloader, optimizer)
    print(f"Training loss: {train_loss}")
    valid_loss, valid_accuracy = evaluate(model, valid_dataloader)
    print(f"Validation loss: {valid_loss}")
    print(f"Validation accuracy: {valid_accuracy}")

# Testing
_, test_accuracy = evaluate(model, test_dataloader)
print(f"Test accuracy: {test_accuracy}") # 정확도 0.82
```

✓ 5m 3.0s

```
Epoch 1/1
100%|██████████| 25/25 [04:20<00:00, 10.40s/it]
Training loss: 1.052655689716339
100%|██████████| 7/7 [00:20<00:00,  2.94s/it]
Validation loss: 1.4215800889900752
Validation accuracy: 0.52
100%|██████████| 7/7 [00:22<00:00,  3.20s/it]
Test accuracy: 0.46
```

3.4.4 학습한 모델 업로드하기

학습한 모델을 나중에 다시 사용할 수 있도록 저장하거나 협업을 위해 공유하는 경우 적용

- 업로드 방법
  - Trainer를 사용한 경우
    - trainer 인스턴스에서 push_to_hub() 메서드 사용하면 학습 모델과 토크나이저를 함께 업로드
  - 직접 학습한 경우
    - 모델과 토크나이저를 각각 push_to_hub() 메서드로 업로드

▼ 예제 3.29 허깅페이스 허브에 모델 업로드

```
from huggingface_hub import login

login(token="본인의 허깅페이스 토큰 입력")
repo_id = f"본인의 아이디 입력/roberta-base-klue-ynat-classification"
# Trainer를 사용한 경우
trainer.push_to_hub(repo_id)
# 직접 학습한 경우
model.push_to_hub(repo_id)
tokenizer.push_to_hub(repo_id)
```

3.5 모델 추론하기

- pipeline을 활용하는 방법
- 직접 모델과 토크나이저를 불러와 활용하는 방법

3.5.1 파이프라인을 활용한 추론

- 허깅페이스는 토크나이저와 모델을 결합해 데이터의 전후처리와 모델 추론을 간단하게 수행하는 pipeline 제공
  https://huggingface.co/docs/transformers/main_classes/pipelines

▼ 예제 3.30 학습한 모델을 불러와 pipeline을 활용해 추론하기

```
from transformers import pipeline

model_id = "본인의 아이디 입력/roberta-base-klue-ynat-classification"

model_pipeline = pipeline("text-classification", model=model_id)
                                                  작업 종류
model_pipeline(dataset["title"][:5])          예측 확률이 가장 높은 레이블과 그 확률을 반환
# [{'label': '경제', 'score': 0.9940265417098999},
#  {'label': '사회', 'score': 0.9847791790962219},
#  {'label': 'IT과학', 'score': 0.9899107813835144},
#  {'label': '경제', 'score': 0.993854284286499},
#  {'label': '사회', 'score': 0.9936111569404602}]
```

3.5.2 직접 추론하기

- 직접 모델과 토크나이저를 불러와 pipeline과 유사하게 추론 구현

▼ 예제 3.31 커스텀 파이프라인 구현

```
import torch
from torch.nn.functional import softmax
from transformers import AutoModelForSequenceClassification, AutoTokenizer

class CustomPipeline:
  def __init__(self, model_id):
    self.model = AutoModelForSequenceClassification.from_pretrained(model_id)
    self.tokenizer = AutoTokenizer.from_pretrained(model_id)
    self.model.eval()

  def __call__(self, texts):
    tokenized = self.tokenizer(texts, return_tensors="pt", padding=True,
truncation=True) ❶

    with torch.no_grad(): ❷
      outputs = self.model(**tokenized)
      logits = outputs.logits

    probabilities = softmax(logits, dim=-1) ❸
    scores, labels = torch.max(probabilities, dim=-1)
    labels_str = [self.model.config.id2label[label_idx] for label_idx in labels.tolist()]

    return [{"label": label, "score": score.item()} for label, score in zip(labels_
str, scores)]
custom_pipeline = CustomPipeline(model_id)
custom_pipeline(dataset['title'][:5])

# [{'label': '경제', 'score': 0.9940265417098999},
#  {'label': '사회', 'score': 0.9847791790962219},
#  {'label': 'IT과학', 'score': 0.9899107813835144},
#  {'label': '경제', 'score': 0.993854284286499},
#  {'label': '사회', 'score': 0.9936111569404602}]
```

모델과 토크나이즈 불러오기

토큰화 수행

모델 추론 수행

가장 큰 예측 확률을 갖는 클래스를 추출해 반환

3.6 정리

참고 자료

- Auto 클래스: https://huggingface.co/docs/transformers/model_doc/auto
- 허깅페이스 허브
  - https://huggingface.co/docs/hub/security-tokens
  - https://huggingface.co/docs/transformers/model_sharing