🧠 BERT Fine-Tuning Lab

DistilBERT로 IMDB 리뷰 감정 분류하기

Machine Learning Study · 2025

실습 소개

이 튜토리얼은 **IMDB 영화 리뷰 데이터셋**을 사용하여 긍정/부정을 분류하는 모델을 만드는 과정을 담고 있습니다. 사전 학습된 `distilbert-base-uncased` 모델을 가져와 우리의 목적에 맞게 **파인튜닝(Fine-tuning)** 합니다.

STEP 1 · 문제 정의
영화 리뷰 텍스트를 입력받아 긍정(1) 또는 부정(0)으로 분류하는 이진 분류 모델을 구축합니다.

STEP 2 · 모델 정보
DistilBERT(경량화된 BERT)를 사용하여 빠르고 효율적인 학습을 진행합니다. (파라미터: 약 66M)

라이브러리 설치

필수 패키지 설치

Hugging Face의 생태계를 활용하기 위해 두 가지 핵심 라이브러리를 설치합니다.

- `transformers` : 사전 학습된 모델(BERT 등)을 쉽게 불러오고 학습시키는 도구
- `datasets` : IMDB 같은 벤치마크 데이터를 한 줄의 코드로 다운로드
- `fsspec` : 파일 시스템 처리를 위한 의존성 패키지

```
# 세션 다시 시작 필요 (코랩 환경일 경우)
!pip install -q transformers datasets
!pip install -U "datasets<=2.18.0" "fsspec<=2023.6.0"
```

데이터 로드 및 분할

IMDB 데이터셋 불러오기

`load_dataset` 함수를 사용해 데이터를 불러옵니다. 이 실습에서는 빠른 실행을 위해 **50개의 샘플**만 사용합니다.

코드 상세 분석:

- `split="train[:50]"` : 전체 학습 데이터 중 앞에서 50개만 슬라이싱
- `.train_test_split(test_size=0.2)` : 8:2 비율로 학습용(40개)과 검증용(10개)으로 분리

```
from datasets import load_dataset

# 1. 데이터 로드
# IMDB 영화 리뷰 데이터셋 중 50개 샘플만 가져와서 학습용/평가용으로 8:2 비율로 나눕니다
dataset = load_dataset("imdb", split="train[:50]").train_test_split(test_size=0.2)
```

데이터 확인하기

데이터가 잘 로드되었는지 확인하기 위해 8번째 샘플(인덱스 7)을 출력해 봅니다.

```
sample = dataset["train"][7]
print(f"📝 리뷰 내용:\n{sample['text']}\n")
print(f"🏷️ 라벨 (0=부정, 1=긍정): {sample['label']}")
```

▸ Output

📝 리뷰 내용: I received this movie as a gift, I knew from the DVD cover... (중략) ... Even if you like B HORROR movies, don't watch this movie

🏷️ 라벨 (0=부정, 1=긍정): 0

라벨 0은 부정적인 리뷰를 의미합니다. 리뷰 내용 마지막의 'don't watch this movie'에서 부정적인 감정을 알 수 있습니다.

모델 및 토크나이저 준비

사전학습 모델 불러오기

우리는 처음부터 영어를 가르치는 것이 아니라, 이미 영어를 잘하는 **DistilBERT** 모델을 데려와서 "리뷰 분류" 업무를 가르칠 것입니다.

Tokenizer
사람의 언어(글자)를 모델이 이해하는 언어(숫자)로 번역해주는 도구입니다.

Model
숫자로 된 언어를 입력받아 분류 결과를 예측하는 두뇌입니다.

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification

# 2. 모델 및 토크나이저 준비
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
# uncased: 대소문자 구분 없이 모두 소문자로 처리한다는 의미

model = AutoModelForSequenceClassification.from_pretrained("distilbert-base-uncased")
# SequenceClassification: 문장 분류를 위한 헤드(Head)가 달린 모델을 불러옴
```

데이터 전처리 (토큰화)

전처리 옵션 설명

- `truncation=True` : 문장이 너무 길면 모델의 최대 길이(512)에 맞춰 자릅니다.
- `padding=True` : 문장이 짧으면 빈 공간을 0으로 채워 길이를 맞춥니다.
- `batched=True` : 한 번에 여러 개씩 처리하여 속도를 높입니다.

```
# 3. 데이터 전처리 (텍스트를 숫자 토큰으로 변환)
def tokenize(batch):
    return tokenizer(batch["text"], truncation=True, padding=True)

dataset = dataset.map(tokenize, batched=True)
```

훈련 설정

Trainer 설정 및 메트릭 함수

Hugging Face의 `Trainer` 클래스를 사용하면 복잡한 학습 루프(for문)를 직접 짤 필요가 없습니다. 학습에 필요한 설정값(Hyperparameters)과 평가 방법만 정해주면 됩니다.

```
from transformers import TrainingArguments, Trainer
from sklearn.metrics import accuracy_score

# 정확도 계산 함수: 모델의 예측값과 실제 정답을 비교하여 정확도를 계산
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = logits.argmax(axis=-1)  # 확률이 가장 높은 클래스 선택
    acc = accuracy_score(labels, predictions)
    return {"accuracy": acc}

# 훈련 파라미터 설정
args = TrainingArguments(
    output_dir="test",           # 모델 체크포인트 저장 경로
    per_device_train_batch_size=8,  # 배치 크기 (한 번에 학습할 데이터 수)
    num_train_epochs=15,         # 학습 반복 횟수 (전체 데이터를 15번 봄)
    report_to="none",            # 외부 로깅 비활성화
```

```
    logging_steps=1              # 매 스텝마다 로그 출력
)

# Trainer 객체 생성
trainer = Trainer(
    model=model,                 # 학습시킬 모델
    args=args,                   # 위에서 정의한 설정값
    train_dataset=dataset["train"], # 학습 데이터 (40개)
    eval_dataset=dataset["test"],   # 평가 데이터 (10개)
    compute_metrics=compute_metrics # 평가 방식 (정확도)
)
```

파인튜닝 실행

이제 모든 준비가 끝났습니다. `trainer.train()` 한 줄이면 학습이 시작됩니다. 모델은 우리가 제공한 40개의 데이터를 15번 반복해서 보며(Epochs), 긍정과 부정을 구분하는 법을 스스로 깨우칩니다.

```
# 5. 파인튜닝 실행 (모델 학습)
trainer.train()
```

▸ Output

```
[75/75 00:34, Epoch 15/15] Step Training Loss 1 0.774700 ... 75 0.001400

TrainOutput(global_step=75, training_loss=0.0416, ...)
```

→ **Training Loss**가 0.77에서 시작해 0.0014까지 떨어진 것을 볼 수 있습니다. 이는 모델이 학습 데이터에 대해 거의 완벽하게 적응했음을 의미합니다.

모델 평가

학습에 사용하지 않은 **테스트 데이터(10개)**를 사용해 모델의 실제 실력을 검증합니다.

```
results = trainer.evaluate()
print(results)
```

▸ Output

```
{'eval_loss': 0.00109, 'eval_accuracy': 1.0, ...}
```

✓ **정확도(accuracy)가 1.0(100%)이 나왔습니다!** (샘플 데이터가 적어서 나온 결과입니다)

실제 예측 수행

새로운 문장 테스트

이제 학습된 모델을 사용해, 본 적 없는 새로운 리뷰 문장을 판별해 봅니다. 이 과정을 **인퍼런스(Inference)**라고 합니다.

💡 **Tip:** 모델은 GPU(cuda)에 있으므로, 입력 데이터도 `.to("cuda")` 를 통해 GPU로 옮겨줘야 에러가 나지 않습니다.

```
# 6. 학습된 모델로 실제 예측 수행
text = "I would put this at the top of my list of films in the category of unwatchable trash!"
# "이 영화를 내 '시청 불가 쓰레기' 카테고리 영화 리스트의 맨 위에 올리겠다!" (혹평)

# 1. 토큰화 및 GPU로 이동
inputs = tokenizer(text, return_tensors="pt").to("cuda")

# 2. 모델 예측
output = model(**inputs)

# 3. 결과 해석 (Logits -> Label)
label = output.logits.argmax(-1).item()
# output.logits는 [부정점수, 긍정점수] 형태입니다.
# argmax(-1)은 둘 중 더 큰 점수의 위치(인덱스)를 찾습니다.

print("긍정" if label == 1 else "부정")
```

▸ Output

```
부정
```

🎉 실습 완료!

이제 여러분은 **BERT 모델을 파인튜닝**하여
감정 분석 모델을 만들 수 있게 되었습니다.