영화리뷰 감성 분석 분류 예측 프로젝트

주제

한국어 영화 리뷰를 기반으로 한 감성 분류 모델을 개발하는 프로젝트이다.

각각의 label은 0(강한부정), 1(약한부정), 2(약한긍정), 3(강한긍정) 으로 이루어져 있다.

목표와 평가 방식

`review` 를 통해 추측한 `label`이 정답 `label` 과 일치하는지의 정확도를 평가 지표로 사용했다.

0,1,2,3의 label이 있고, 각각은 독립적인 label로 취급해 label이 맞는지, 틀린지만 검증했다.

데이터셋

학습 데이터셋 279650개, 평가 데이터셋 59928개가 주어졌다.

학습 데이터셋의 절반은 증강된 데이터로, 기존 데이터를 이용하여 만든 데이터셋이다.

학습 데이터셋은 각각 데이터셋의 id인 `id`, 리뷰 `review`, 원본인지 증강 데이터셋인지를 알리는 `type`, 0부터 3 사이의 정수 값을 갖는 `label`로 이루어져 있다.

평가 데이터셋은 `id` , `review` 로만 이루어져 있다.

조건과 환경

평가 데이터를 Pseudo-labeling하거나, 눈으로 판별 후 라벨링은 불가능하다.

다만, TAPT를 통한 사전학습은 허용되었다.

외부 데이터셋의 활용은 불가능했지만, 직접 Augmentation 혹은 LLM 모델을 이용한 합성 데이터 추가는 가능하다.

사전 학습 가중치는 `klue/roberta.base` , `klue/bert-base` , `kykim/bert-kor-base` , `bemoi/kcbert-base` , `monologg/koelectra-base-ve-discriminator` 5개를 사용 가능했다.

서버는 NVIDIA Tesla V100 GPU를 사용하였으며, VScode에 연결하여 작업을 진행하였다.

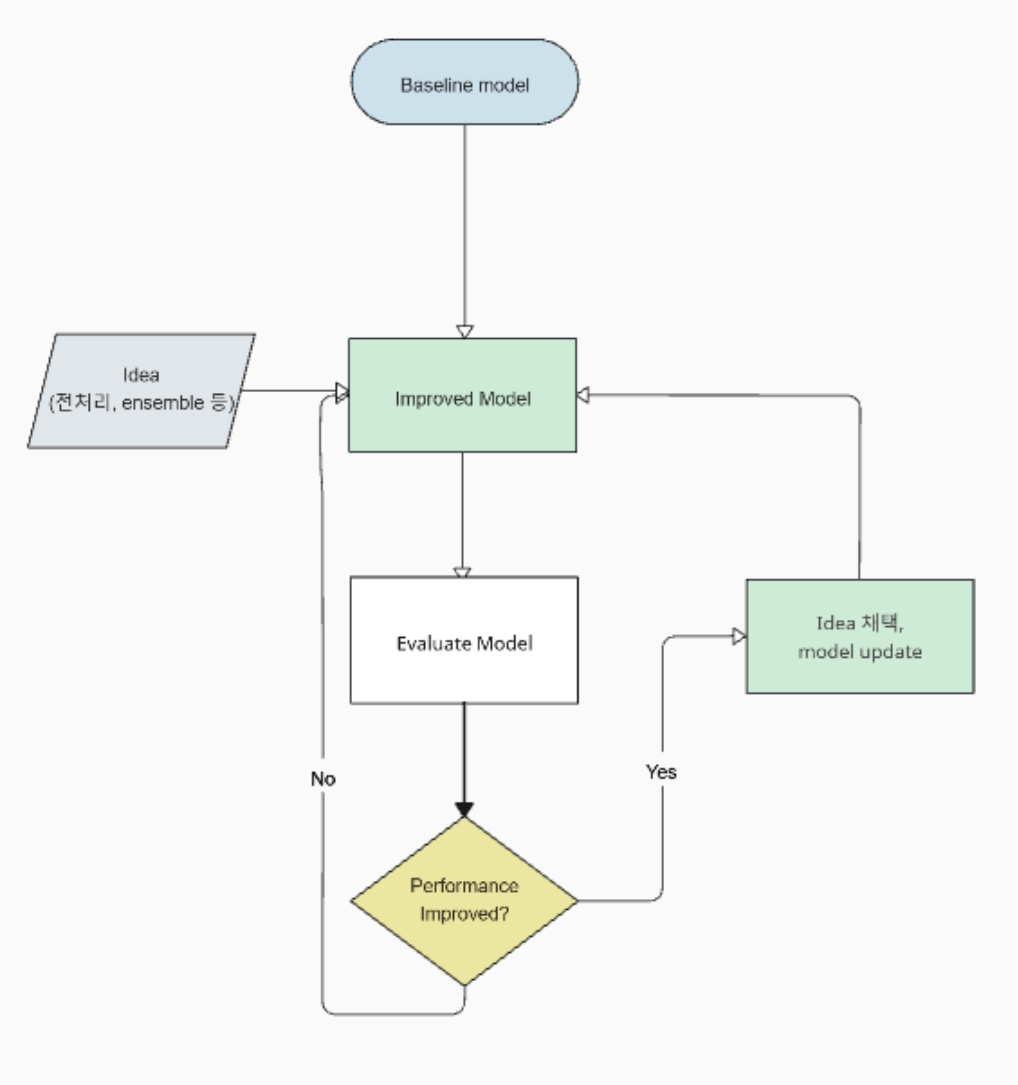
저장 공간은 약 50GB가 주어졌으며, 그 중 기본 환경 설정을 위한 용량을 제외하면 약 36GB 정도를 학습 중간 모델 저장 등에 사용할 수 있었다.

일정

2025.10.22 10:00 ~ 2025.10.30 19:00

수행 절차

수업을 먼저 전부 이해하고 나서, 2025.11.25부터 프로젝트를 시작하였다.



다음과 같은 과정을 반복하며, 모델을 개선해 나갔다.

Baseline 코드 이해 (2025.11.25)

Baseline 코드는 전처리 → transformer의 trainer를 통한 학습 → inference 순서로 이루어졌다.

모델은 `klue/bert-base` 를 사용하였으며, 10%의 validation dataset에 대해 Accuracy 0.8503을 기록하였다.

이를 기반으로 하여, 하나의 `ipynb` 파일을 여러 개의 역할을 하는 `trainer.ipynb` , `dataset.py` , `preprocess.py` , `functions.py` 로 분리하여 파일 관리를 쉽게 하였다.

실행 계획

먼저 Hyperparameter의 경우에는 train batch size = 256 , learning rate = 2e-5, num_epochs = 5, warmup_steps = 500, weight_decay = 0.01을 기본 setting으로 하고 진행하였다.

Early Stopping을 활용하여, Validation Accuracy가 증가하지 않는 경우에는 학습을 멈추게 하였다. patience=2로 설정하여, 한 번의 여유를 주었다.

전처리

Baseline 코드를 기반으로 하여, 텍스트 전처리에서 불완전한 한글 제거를 하는 부분에서 ㅎㅎ, ㅠㅠ 등 "의미가 있는" 초성들을 남기고, 반복되는 초성 정규화 규칙을 완화하였다. 기존 정규화 규칙은 2개가 넘는 초성들을 2개로 줄이는 방식이었는데, 한국어에서 "ㅋㅋ"와 "ㅋㅋㅋ"처럼 2개의 초성과 2개가 넘는 초성은 그 의미가 또한 다르다고 판단해 초성이 3개까지 유지되게 하였다.

과제에서 배운 대로, 링크나 메일 주소 등은 [URL], [MENTION], [EMAIL] 으로각각 정규화하여 저 문자들로 만들어 준 뒤, 특수 토큰으로 만들어, `tokenizer`에 추가해 주었다.

Ensemble

사용 가능한 다섯 개의 사전학습된 모델들을 전부 사용하기 위해, 5개의 모델을 Ensemble하고, 다섯 개의 모델들의 logit을 이어붙혀 하나의 선형 레이어를 stacker로 사용하여 통과시켜 라벨들에 대한 logit이 나오게 하였다.

Stacker 학습은 Cross-entropy loss를 사용하였는데, epoch이 큰 학습의 특성 때문인지 실행 시마다 결과가 다르게 나오는 문제가 있었다. `set_seed()` 관리가 제대로 되지 않은 탓으로 보인다.

각각의 모델을 사용한 것보다 성능은 좋게 나왔지만, 학습 시간이 5배가 걸렸기 때문에 여러 가지 방법론을 시도하기 힘들어지는 원인이 되기도 하였다.

AdaBoost

AdaBoost는 Ensemble 전의 각각의 5개 모델들에 대해 진행하였다.

`trainer`의 `Callback` 을 이용하여 다음 모델에 대한 α 와 sample weight을 사용하여, 더 어려운 데이터셋을 학습하는 learner 여러 개를 만들어 ensemble하기 전에 합치게 되었다.

하지만, 이 경우에는 오히려 5개의 모델 전부, 그리고 그 합친 모델까지 모두 accuracy와 loss가 악화되는 현상을 보여 AdaBoost는 폐기하게 되었다.

TAPT

사전학습된 모델 5개를 다시 Task에 맞추어 pretrain을 먼저 진행하였다. (label을 통한 학습 전에)

Train dataset뿐만 아니라 Test Dataset을 포함한 pretrain을 진행하였다.

15%의 단어를 비우고 비운 단어를 맞추는 Task를 통해 TAPT가 완료된 5개의 모델을 만들었으며, 이후에는 이를 기반으로 학습을 진행하였다.

Optuna를 이용한 Hyperparameter Tuning

Optuna를 이용하여, Hyperparameter Tuning을 진행하였다.

여러 가지 경우의 수를 고려하기 위해, epoch = 10으로 충분히 크게 해 놓은 뒤 early stopping을 걸어 놓고,

```
hyperparameter_space = {
        "learning_rate": (1e-5, 5e-4),
        "per_device_train_batch_size": [ 64, 128, 256],
        "weight_decay" : (0.005, 0.05)
}
```

으로 Tuning하였다.

Regressor로의 전환

앞의 모델들은 multi-label classification 문제를 푸는 모델이다. 하지만, 감성 분류 같은 경우에는 "약한 긍정"을 "강한 긍정"으로 잘못 읽는 것과 "강한 부정"으로 잘못 읽는 것에 대해 같은 loss를 주는 것이 적절하지 않다고 판단하여,

Bert 모델을 회귀 모델로 만들어 logit을 뱉은 뒤, (0,3)으로 clipping한 후 (loss를 구하고) round하여 결과를 뱉는 모델을 만들어 보았다.

learning rate의 경우에는 위에서 Optuna를 이용하여 나온 최적의 hyperparameter를 사용하였고, learning rate만 회귀 모델에 맞추어 0.5를 곱해 주어 사용하였다.

학습에 시간이 모자라, 원하는 성능까지 끌어내지는 못하였다.

결과

최종 제출에는 TAPT를 적용한, Optuna를 통해 최적화한 hyperparameter로 학습시킨 5개의 모델을 Ensemble한 모델 중, Validation Accuracy, Validation Loss를 기준으로 최적의 모델을 고른 (bert에서) 두 모델을 선정하여 제출하였다.

Accuracy 0.8325로, 81등. 만족스럽지는 않은 결과가 나왔다.

가장 시간을 많이 쏟은 AdaBoost에서 성능이 나오지 않았고, 계획 순서로 인해 학습 시간이 길어진 것이 문제로 보인다.

평가

시간 관리의 문제점

TAPT 같은 경우는 label에 대한 "학습 전"에 이루어져야 한다.

하지만, TAPT를 하기 전에 먼저 TAPT 없이 여러 가지 학습을 진행하였고, TAPT를 쓰기로 결정한 이후에는 (학습을 진행한) 모델을 다시 사용하지 못하고 전부 폐기해야 했다.

또한, 평가 Metric 혹은 hyperparameter, 저장 등에 문제가 생길 수 있어 미리 `df.sample` 을 통해 작은 dataset에 대해 코드가 잘 돌아가는지 확인하였는데, 이 부분에서 확인하지 못한 부분이 많아 절반 정도 진행되었던 학습을 폐기해야 하는 경우가 많았다.

Random과 Ipynb

메인 학습과 stacker을 이용한 ensemble을 Ipynb에서 진행하였는데, 이 과정에서 random seed에 대한 이해 부족으로 재현성에 대한 문제가 발생하였다.

Optuna를 통한 학습을 할 때, 최적의 hyperparamter로 “재학습”을 진행하였는데, 이 과정에서 random 값이 달라져 optuna hyperparameter tuning 시와는 다른 결과를 뱉는 문제가 있었다.

또한, Ipynb cell을 여러 번 실행하는 등의 behavior가 문제가 되었다. random_seed를 그때마다 초기화해 주는 등의 추가 설정이 부족했고, 따라서 재현성에 문제가 생겼다.

회고와 개선 방향

학습에 걸리는 시간을 잘 고려하지 않은 계획과, 잘못된 코드를 실행시켜 코드를 중단하느라 버린 시간이 너무 많았다.

특히 마지막 Regressor로 문제를 바꾸어 보는 것은 학습이 느려 긴 시간이 필요했는데, 5개 모델을 전부 제대로 학습시키지 못하고 따라서 optuna를 통한 hyperparameter tuning도 하지 못하였다.

주어진 시간과 학습에 걸리는 시간을 고려한 철저한 계획과 , 코드가 잘 돌아가는지 정확하게 확인 후에 학습을 진행하는 태도가 필요함을 느꼈다.

또한, 재현성을 위해 고려해야 할 요소들에 대해 생각해 보게 되었다. 단순히 코드 시작에 Random_seed를 설정한 경우, Ipynb에서 cell을 여러 번 실행하는 등의 행동의 문제점을 조심해야겠다.