기업과제3 NLU - 문장 유사도 계산 (STS)

기간 : 2022-03-16 ~ 2022-03-19

팀원 : 남준우, 조성빈, 권예환, 강상훈, 장수림

개요

- 목표
  - klue-STS 데이터를 이용하여 의미적 텍스트 유사도(Semantic Textual Similarity) 모델을 훈련
  - 가장 성능이 좋은 학습 방식 및 모델을 발견하고 그것들을 토대로 모델 제작 및, API 구성
- 데이터 셋
  - klue-STS
    - klue-STS는 AIRBNB(리뷰), Policy(뉴스), paraKQC(IoT기기 질의문)등의 문장 데이터를 모아두었으며 문장 - 문장의 페어를 이루어져 있으며 각 문장간의 유사도를 0 ~ 5 까지 점수를 매겨둔 데이터 셋 입니다.
    - train셋과 dev셋, Test셋 으로 구분 되있으며 개수는train 셋 11688개 이며, dev 519개, Test셋 1037개의 갯수를 가지고있습니다.
- 진행 시 주의사항
  - Train과 Dev 데이터가 공개되어 있습니다. Train set만을 사용하여 모델을 훈련해 주세요.
  - Train set을 다시 training data와 validation data로 나누어서 훈련시키는 것을 권장합니다.
  - 모델은 어떤 NLP 모델을 사용해도 됩니다. BERT 등 주어진 학습 환경에서 훈련 가능한 모델을 활용해주세요.
  - 공개된 Pretrained 모델을 사용하여도 됩니다. (출처 명시)
- 과제 결과물

- 학습된 모델 (모델 자유 선택)
- 학습 방식 보고서
  - 자신이 담당한 역할 (ex. 모델링, 논문 리서치, 하이퍼파라미터 튜닝 등) 설명
  - 어떤 모델을 선택했나
  - 어떻게 파라미터를 튜닝했나
  - 어떤 훈련 과정을 거쳤는가
  - 최종 결과 분석
- dev set score (dev set의 모든 문장 pair에 대한 유사도 추론 결과와 F1 점수)
- REST API를 통해 모델을 이용하여 두 문장의 유사도를 분석하는 Server Code

팀 전략

- 저희 팀의 경우 각 인원을 전처리 전담 인원 과 API전담 인원으로 분리하여 작업을 진행하였으며 학습 모델 및 방법론의 경우 모든 팀원이 각자 자신만의 전략과 모델을 가지고 학습을 진행하였으며, 그 중 가장 성능이 괜찮았던 모델을 API로 만들기로 하였으며 전처리 전담 인원과 API전담 인원은 아래와 같습니다.

- 전처리 전담 인원
  - 조성빈, 장수림, 권예환
- API 전담 인원
  - 남준우, 강상훈

진행 과정

- 전처리
  - 전처리는 기본적으로 전처리 전담 팀 에서 진행하였으며 제 모델에 맞게 제가 조금씩 변형하였으며, 요청 한 사항도 있었습니다.
  - train데이터 셋을 학습을 위해 train과 valid를 9:1로 나누어서 학습에 사용하였습니다.
  - Scaled cosine similarity의 경우에는 필요한 데이터인 pair를 이루는 문장과 real-label을 가져와서 스케일링 (÷5)를 진행하여 0 ~ 1사이로 나타낼 수 있도록 soft-

labeling을 진행 하였습니다.

- element-wise product와 siam-network 파생 모델의 전처리의 경우 전처리 전담 팀에서 전처리를 진행 해 주었으며 label중 가장 많이 나온 label을 선정하여 real-label로 지정해주었습니다.(Softmax 사용을 위해)

  > https://s3-us-west-2.amazonaws.com/secure.notion-static.com/f8c5384d-fafc-4be9-8404-455f8f61a44d/dataset.csv

- 이후 klue/bert-base Tokenizer를 이용하여 Tokenizing을 진행하였으며 Tokenizer 파라미터는 다음과 같습니다.
  - add_special_tokens = True
  - padding = 'longest',
  - truncation = True
  - return_tensors= "pt”

- 학습 모델
  - klue/bert-base 모델
    - 모델 소개
      - Klue/Bert-Base는 KLUE(Korean Language Understanding Evaluation)에서 제작한 Bert의 대표적인 한국어 Pre-trained 모델이며, 여러 한국어 데이터를 학습 시켜 약 110M의 파라미터를 가지고 있는 모델입니다.
  - 이 모델을 선택한 이유
    - KLUE 데이터 셋에 맞게 KLUE에서 제공하는 Klue/Bert-Base 모델을 사용하는 것이 알맞다고 생각했습니다. 또한 이번 프로젝트를 진행하면서 KLUE에서 기본적으로 제공해 주는 Baseline에 대해서는 알고 있으나, 저는 이번 프로젝트를 통해 Bert 모델의 사용법을 더 익혀보고자 Baseline을 참고하지 않고 저의 생각을 담아서 진행하였습니다.
- 학습 전략 및 방법론
  - Scaled-Cosine Similarity

- 첫 번째로 접근 한 방식은 코사인 유사도를 이용 해 보는 것이였습니다. 코사인 유사도의 경우 계산 시 -1~1 사이로 문장 간의 거리를 계산해줍니다. 이런 계산을 할 input값들은 bert모델을 거쳐서 나오는 pooler를 통하여 계산이 가능한데, 현재 답으로 주어진 라벨의 경우 0~5까지의 정수로 이루어져있고 이것을 스케일링(÷5)를 하여도 0~1사이의 실수로 이루어지며 사실 상 코사인 유사도를 이용한 분석이 불가능해 보일지도 모르지만 코사인 유사도를 0 ~ 1사이로 스케일링을 해준다면 유사도 계산을 이용하여 학습이 가능해 질 것이라 판단하여 코사인 유사도를 스케일링 후에 학습을 진행하였습니다.

  https://s3-us-west-2.amazonaws.com/secure.notion-static.com/4f7d2b22-c149-4818-8416-418173148dc4/corp3_Preprocessed_scaled_cosine_similarity.ipynb

  - 스케일링 식은 간단하게 (cosine_similarity + 1) / 2 로 진행하였습니다.
- Element-Wise Product
  - 두 번째 접근 방식은 element-wise product입니다. Attention is all you need에서 나온 self-attention 에서 각 Q, K, V에서 Attention Value를 뽑아 낼 때 {(Q, K.T)/(dk ** 0.5)}V에서 (Q, K.T)를 Dot-Product를 하는 방식에서 착안해왔습니다. Self-Attention에서 Q, K, V를 Product 함으로써 서로의 연관성을 텐서화 하는 과정이 있는데, 이러한 방식을 채용하여 Element-wise product를 통하여 서로의 연관성을 계산 후에 softmax를 통하여 유사성을 뽑아낼 수 있을까 생각하여 이러한 방식에 대한 시도를 진행하였습니다.

    https://s3-us-west-2.amazonaws.com/secure.notion-static.com/c121c66a-39e5-4c6d-974d-bc78561a98d6/corp3_Preprocessed_element_wise_product.ipynb

    - Dot-product가 아닌 이유
      - Attention의 경우 dot product를 진행하였으나, 현재 이번 프로젝트의 pooler의 경우 전치(Transpose)를 시킬 경우 그 pooler의 의미를 잃어 버릴 것 같았습니다.(순서 쌍이 흐트러짐, pooler의 output은 sequential함) 그래서 최대한 전치를 시키지 않는 선에서 product를

할 방법을 생각했고 최종적으로 element-wise product를 사용하게 되었습니다.

- siam network 파생
  - 세 번째 접근 방식은 siam network 파생 구조입니다. 아래의 그림과 같이 각각의 문장을 풀링 후 abs(u-v)를 진행하고 softmax classifier를 지나는 방식으로 forward를 만들어 학습을 진행하였습니다. 이와 같이 각 두 문장을 별도로 pooling을 진행하는 모습이 마치 siam network와 비슷하여 이름을 siam network 파생이라고 이름을 붙였습니다.

    https://s3-us-west-2.amazonaws.com/secure.notion-static.com/4bab12ca-a8c8-4edc-a35a-eb38bc62ff53/corp3_Preprocessed_siam_network.ipynb

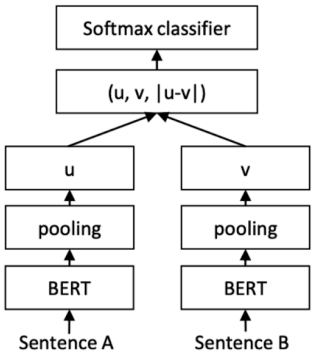


Figure 1: SBERT architecture with classification objective function, e.g., for fine-tuning on SNLI dataset. The two BERT networks have tied weights (siamese network structure).

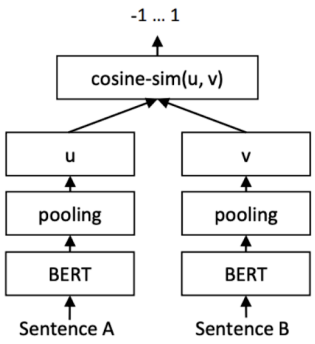


Figure 2: SBERT architecture at inference, for example, to compute similarity scores. This architecture is also used with the regression objective function.

SBERT의 논문의 아키텍쳐 설명

- 하이퍼파라미터 설정
  - 하이퍼파라미터 튜닝의 경우 휴리스틱 하게 진행하였으며, 최종적으로 튜닝 된 파라미터는 아래와 같습니다.

    epoch = 100(Scaled-Cosine Similarity = 50),

    dropout rate = 0.1

    Learning rate = 1e-4(0.0001)

max grad norm = 1 #for gradient clipping

weight decay = 0.01

loss = Cross entropy(element-wise product, siam network 파생),

MSE(Scaled-Cosine Similarity)

optimizer = AdamW

결과

- 나의 모델 결과
  - siam network 파생
    epoch 100 train acc 0.5001585489599188
    epoch 100 test acc 0.4639187866927593
    dev f1 score 0.5221579961464354
    dev corrcoef 0.20050435

  - element-wise product
    epoch 100 train acc 0.5492135971588026
    epoch 100 test acc 0.5024461839530333
    dev f1 score 0.4816955684007707
    dev corrcoef 0.123599

  - scaled-cosine similarity
    epoch 50, train MSE loss 0.0014963836874812841
    epoch 50, test MSE loss 0.10389275848865509
    dev f1 score 0.6622734761120265
    dev corrcoef 0.33085414

- 최종 모델 선정 결과
  - 최종적으로는 저희 팀에서 가장 F1 Score가 높았던 장수림 팀원님의 모델이였던 Sentence Bert(SBERT)를 이용한 모델을 이용하기로 결정하였습니다.
  - 최종모델 결과
    - SBERT(batch_size=16)

- Pearson'r : valid 0.960 / dev 0.892
- F1 score : dev 0.834

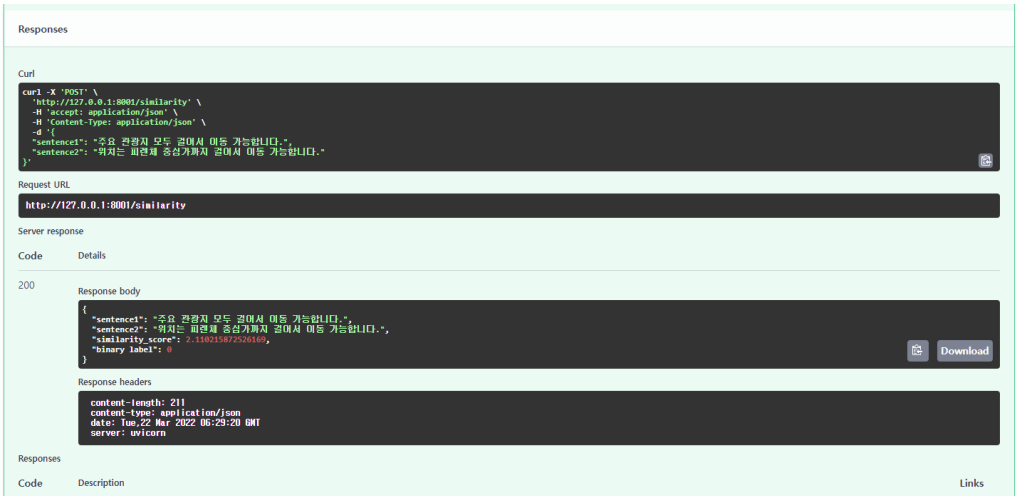
API 제작

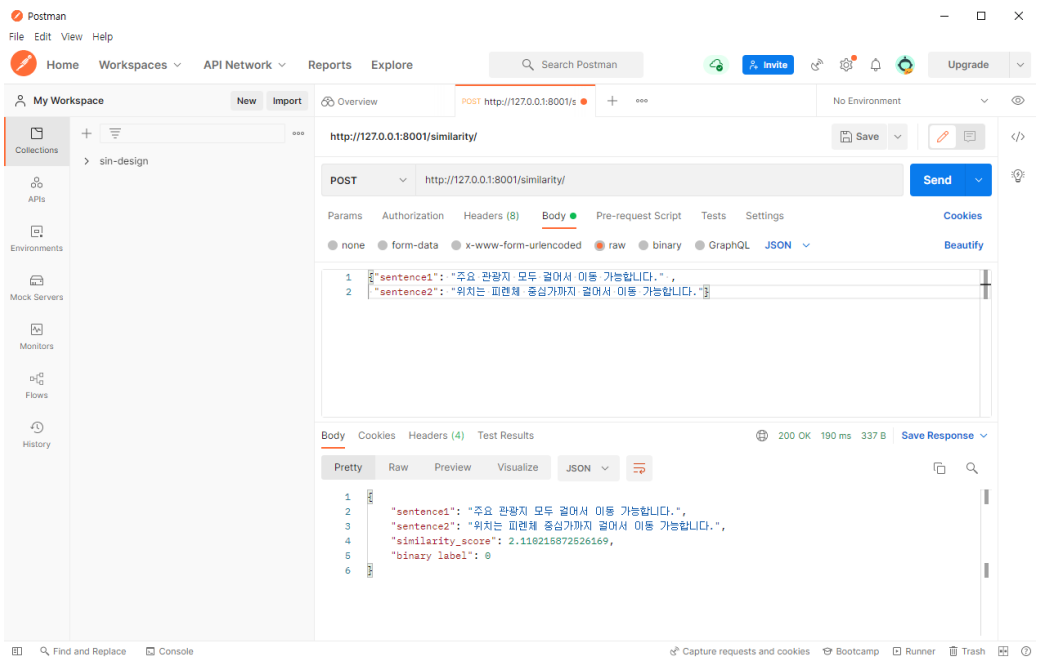
- FastAPI
  - 이번 API 프레임워크는 FastAPI를 사용하였으며 FastAPI의 특징은 다음과 같습니다.
    - **빠름**: (Starlette과 Pydantic 덕분에) **NodeJS** 및 **Go**와 대등할 정도로 매우 높은 성능. 사용 가능한 가장 빠른 파이썬 프레임워크 중 하나.
    - **빠른 코드 작성**: 약 200%에서 300%까지 기능 개발 속도 증가. *
    - **적은 버그**: 사람(개발자)에 의한 에러 약 40% 감소. *
    - **직관적**: 훌륭한 편집기 지원. 모든 곳에서 자동완성. 적은 디버깅 시간.
    - **쉬움**: 쉽게 사용하고 배우도록 설계. 적은 문서 읽기 시간.
    - **짧음**: 코드 중복 최소화. 각 매개변수 선언의 여러 기능. 적은 버그.
    - **견고함**: 준비된 프로덕션 용 코드를 얻으십시오. 자동 대화형 문서와 함께.
    - **표준 기반**: API에 대한 (완전히 호환되는) 개방형 표준 기반: OpenAPI (이전에 Swagger로 알려졌던) 및 JSON 스키마.
- 구성
  - 구성은 inference.py 모듈과 더불어 FastAPI가 올라가는 main.py로 구성되있습니다.
  - inference.py
    - inference.py는 유사도 계산 기능을 모듈화한 py파일이며 이 모듈 안에서 유사도 계산이 이루어집니다.
  - main.py
    - main.py는 FastAPI를 로딩하며 Uvicorn에 올리는 과정을 수행하며. 모델을 initialize를 하는 과정을 진행합니다.
  - 호출 방식
    - Method는 POST방식이며 값은 Json의 형태로 두 문장을 파라미터로써 받습니다.

- json의 형태로 {"sentence1": sentence1, "sentence2": sentence2} 로 API를 호출합니다.

◦ 결과

- github 주소 : https://github.com/namjunwoo223/wanted_pre_onboarding/tree/main/기업과제/GIUP3
- 결과는 json {"sentence1": sentence1, "sentence2": sentence2, "similarity_score" : float, "binary label": 1 or 0}의 형태로 Return을 받습니다.
- Example(Swagger, Postman)





Reference

- SBERT Paper : https://arxiv.org/pdf/1908.10084.pdf
- FastAPI : https://fastapi.tiangolo.com/ko/
- KLUE-Benchmark: https://klue-benchmark.com/