KLUE 문장 내 개체간 관계 추출 랩업 리포트

최재하\_T1215

<기술적인 도전>

LB 점수 0.7900, 72등

* Validation 전략

train 데이터를 train set 0.8, validation set 0.2로 나눠 검증을 하였습니다.

* 사용한 모델 아키텍처 및 하이퍼 파라미터

1. 아키텍처 : XLM-Roberta-large

* + 1. LB 점수 : 0.79
    2. Training time augmentation : 적용하지 않음
    3. batch\_size : 32
    4. tokenized\_max\_lenght : 256
    5. optimizer : AdamW
    6. scheduler : cosine\_with\_restarts\_with\_warmup
* 앙상블 방법

1. 앙상블을 사용하지 않고 단일 모델만을 사용해 제출하였습니다.

* EDA
  + 1. 데이터의 불균형이 큰 것을 알 수 있었습니다. 특히 소수 Label에 대해서는 10개 미만의 적은 데이터 수를 보이는 것이 있었습니다. 소수 데이터에 대해 data augmentation이나 외부 데이터를 가져오는 방법을 생각해 볼 수 있을것 같습니다.
    2. 일부 데이터에 대해 잘못된 label이 할당 되어 있는 것이 발견되었습니다. outlier에 대한 대처가 필요하다고 생각합니다.
    3. Entity의 상당 부분이 고유명사인 것이 확인되었습니다. 고유명사는 token화가 어렵다는 점, 정보를 얻기 힘들다는 점에서 고려해야 할 부분이 많다고 생각합니다.
    4. 한자 데이터도 다수 발견되었습니다. 사전학습된 모델이 한글 데이터 만을 사용했을 경우, fine-tuning이 잘 되지 않을 수 있다고 생각합니다. 다국어 학습 모델이 필요할 수 있습니다.
* 추가 시도, 그에 대한 고찰
  + 1. Automatic Mixed Precision
       - 1. 처리 속도를 위한 FP16연산과 정확도 유지를 위한 FP32 연산을 섞어 학습하는 Automatic Mixed Precision을 적용하였습니다.
         2. 학습 속도가 약 4배 정도 빨라졌기 때문에, 더 많은 실험을 할 수 있었습니다.
         3. 성능이 약 0.2% 떨어졌으나, 이는 오차 범위 내에서 비슷한 성능을 낸다고 생각하였습니다.
    2. label\_smoothing
       - 1. EDA를 통해 학습 데이터 내에 mislabeling 된 데이터가 다수 존재한다는 것을 알았습니다.
         2. 이러한 문제를 해결하기 위해 label\_smoothing을 적용하여 약 1%의 성능 향상을 이루어냈습니다.
    3. cosine\_with\_restarts\_with\_warmup
       - 1. learning rate를 바꾸어 가며 실험을 해 보았을때, learning rate가 모델 성능에 큰 영향이 있는것을 발견하였습니다.
         2. 빠르고, 안정적인 수렴을 위해 warmup을 포함한 cosine\_with\_restart scheduler를 사용하였습니다.
         3. 실험을 통해, warmup step을 300으로, cycle을 3으로 주었을때, 약 2%정도의 성능 향상을 얻어냈습니다.

* + 1. max\_length 변경하기
    2. token화 한 문장의 최대 길이를, 모든 문장이 충분히 담기도록 크게 설정하는 방법과, 최대 길이를 작게 하여, 임의의 길이로 자르는 방법 모두를 고려하여 실험하였습니다.
    3. 최대 길이를 길게 설정하면, 문장의 모든 정보를 담을 수 있지만, 짧은 문장의 경우 PAD 토큰이 많이 들어가, 문장의 정보가 옅어지는 영향이 있을 수 있다고 생각합니다. 또한 gpu메모리 한계상, 작은 batch\_size를 사용해야 하기 때문에, 학습시간이 길어지는 단점이 있고, 성능에 영향을 끼칠 수 있습니다.
    4. 최대 길이를 작게 설정하면, 큰 batch\_size를 사용할 수 있어, 학습시간을 단축 시킬 수 있으나, truncation을 통해 문장의 일부가 잘려나가 정보 손실이 있을 수 있다는 단점이 있습니다. 특히, 문장 내에서 Entity가 뒤쪽에 존재하여 잘려 나갈 경우, 학습하는데 치명적일 수 있다고 생각합니다.
    5. 여러 실험을 통해, 최대 길이를, train dataset의 토큰화된 문장 중 가장 긴 문장의 길이로 맞추는 것이 성능이 제일 높았습니다.
    6. 이는, 문장의 정보 손실을 최소화하면서, 적당히 큰 사이즈의 batch\_size를 사용 할 수 있었기 때문에, 가장 좋은 성능을 얻어낼 수 있었다고 생각합니다.
    7. xlm-roberta-large
       - 1. koelecetra도 구조적으로 좋은 전략을 가진 모델이기는 하나, 모델 사이즈가 충분히 크지 않다는 점, 한국어 데이터만 사전학습을 했다는 점이 아쉬웠습니다.
         2. 따라서, 모델이 아주 크고, 다국어의 상당히 많은 데이터를 사전학습한, xlm-roberta-large로 모델을 변경하였습니다.
         3. 이를 통해, 약 5%의 큰 성능 향상을 얻을 수 있었습니다.
    8. Cross Validation
       - 1. 여러 실험을 적용한 모델에 대해, 최종 성능 향상을 위해 k-fold Cross Validation을 적용하였습니다.
         2. 5개의 fold를 나눠 inference 한 후, 얻은 label을 hard voting을 통해 최종 결과를 얻어냈습니다.
         3. 이를 통해, 약 1%의 성능 향상을 이루어냈습니다.
* 시도했으나 잘 되지 않았던 것들, 그에 대한 고찰
  1. 음절 단위 tokenizer
     1. Wordpiece tokenizer가 고유 명사를 분해 해서, 잘못된 token을 만드는 데서 착안해, 음절 단위 tokenizer를 사용하여 학습하였습니다.
     2. 그러나 성능이 약 2%정도 하락하였습니다.
     3. 이는, 음절단위로 분해함으로써, 단어의 의미 정보 또한 손실되기 때문이라고 생각합니다.
     4. 고유 명사를 잘못 분해하는 경우가 있더라도, 단어의 의미를 보존 하는 것이 더 많은 정보를 담게 될 것이라는 생각을 하게 되어, 추 후, Wordpiece tokenzier를 사용하였습니다.
  2. NER을 사전학습한 Koelecetra 모델
     1. Enitity 간의 관계 추출에 있어서, 그 관계 정보가 NER분류 task와 밀접한 관련이 있을 것이라는 생각에, NER 분류로 사전학습 된 koelectra 모델을 사용하여 fine-tuning을 적용해 보았으나, 약 2%의 성능하락이 있었습니다.
     2. NER을 추출하는 task와, fine-tuning에서 고안한, 문장에 Entity를 붙여 단일 문장 분류하는 task와는 구조가 상이하다는 점, NER label과 우리의 task의 관계 label이 많은 차이가 있다는점 때문에 성능이 좋지 못하였다고 생각합니다.
     3. NER정보를 관계 추출 task에 적용하기 위해, 입력 형태를 바꾸는 등의 연구가 필요하다고 느꼈습니다.
  3. Optimizer, scheduler, Learning rate 변경하기
     1. Adam, SGD 등의 Optimizer와 Linear, StepLR등의 scheduler, 다양한 learning rate를 사용하여 실험을 하였습니다.
     2. 실험결과 AdamW, cosine\_with\_restart\_with\_warmup을 사용하고 1e-5의 learning rate를 사용하였을 때, 가장 높은 성능이 나왔습니다.
     3. 이 중에서, 성능에 가장 큰 영향을 끼쳤던 것이 learning rate였습니다. 너무 높은 learning rate를 사용할경우 학습이 전혀 진행되지 않는 것이 발견되었습니다.
     4. Learning rate는 학습시간에만 큰 영향이 있을 것이라고 예상하였지만, 모델 성능에도 예상외로 큰 영향이 있을 수 있다는 것을 깨달았습니다.
     5. Hyperparameter tuning에 있어서도 많은 노력을 들여야 한다는 것을 느꼈습니다.
  4. 소수의 label 정보를 없애기
     1. Train dataset의 일부 label에 대하여 그 수가 10개 미만인 데이터가 존재하였습니다.
     2. 이는, 학습하기에 충분하지 못한 데이터 수 이기 때문에, 학습에 혼란을 주는 outlier로 고려할 수 있다고 생각하였습니다. 또한 Cross Validation을 하기에는 더욱 부족한 데이터로 판단하였습니다.
     3. 그리하여, 소수의 데이터 수의 label들을 모두 관계없음 label로 바꾸어, train dataset을 새로 구성하는 방법을 적용하였습니다.
     4. 그러나, 약 1%정도의 성능 하락이 있었습니다.
     5. label을 살펴보면, 단체:모회사와 단체:자회사, 인물:거주\_국가와 인물:사망\_국가 등, 일부 label들 사이에는 어느정도 상관관계를 갖는 비슷한 정보를 가지는 것을 알 수 있습니다. 소수 데이터의 label을 관계 없음으로 바꿈으로 인해, 이러한 정보의 손실이 일어나는 것이 아닐까 추측합니다.
  5. NER Tag 활용하기
     1. 모델에게 Entity의 관련 정보(NER)을 추가로 주는 방법을 통해, NER과 관련성 높은 관계 추출 task를 더 잘 풀 수 있지 않을까 하는 접근으로, 사전학습된 NER태그를 Entity 앞 뒤에 붙여줌으로서 적용하였습니다.
     2. 하지만, 모델 성능이 약 1~2%하락하는 결과를 얻었습니다.
     3. Train dataset의 Entity의 NER을 추출해 살펴본 결과, 사람이름 ‘새봄’이 시기를 나타내는 ‘DATE’로 표현되는 등의, 미스매칭이 다수 존재한다는 것을 알았습니다.
     4. Train dataset의 특성상, 관계 추출에 이용되는 Entity가 대부분 고유명사인 점에서, 올바른 NER 추출이 되지 않아, 오히려 학습에 방해가 되는 것이 성능하락의 이유라고 생각합니다.
     5. NER은 관계 추출 task와 밀접한 연관성을 보이고, NER tag를 붙이는 방법 또한 여러가지 방법이 존재하므로, NER을 활용하는 방안에 대한 연구가 더 필요하다고 생각합니다.
  6. Knowledge Graph 활용
     1. 관계 추출 task에 있어서, 사전에 구성된 Knowledge Graph에서 정보를 가져와서 적용하면 더 좋은 성능을 낼 수 있을 것이라고 생각하였습니다.
     2. 그러나 knowledge graph를 만드는 것은 많은 노력이 필요하고, 사전 제작된 knowledge graph는 대부분 영어 데이터로 구성되어 있기 때문에, 적절한 데이터를 찾는 것이 어려웠습니다.
     3. 일부 한국어 데이터를 포함한 graph set을 찾을 수 있었지만, 구현이 난해하여, 본 task에 적용하는 것에 실패하였습니다.
     4. 관계 추출 task에 knowledge graph를 활용하는 방법은, 좀 더 깊은 연구가 필요하다고 생각했습니다.

<학습과정에서의 교훈>

* + 1. Stage1 의 경험을 바탕으로 나만의 개성있는 전략을 세울 필요가 있었습니다. 이번 Stage에서는 Knowledge Graph활용이라는 전략을 구성 하였으나, 결과적으로 구현은 실패하였습니다. 부족한 논문 이해 능력과, 구현력으로 인해, 적용하지 못하였다고 생각합니다. 더 많은 논문을 읽어보고 구현해보는 연습을 해야겠다고 생각했습니다.
    2. 저번 Stage를 통해 모델 선택, 하이퍼파라미터는 중요도가 낮고, 번뜩이는 전략들이 더 중요하다고 느꼈었는데, 이번 Stage를 통해, 같은 모델, 같은 전략을 사용하더라도, 사소한 차이에 의해 성능의 큰 차이가 발생할 수 있다는 것을 알았습니다. 사소한 값이라도 깊은 고찰을 통해 구상을 하고, 실험을 진행해 보아야겠다고 생각했습니다.
    3. 다시한번 시간 관리가 매우 중요하다는 것을 깨달았습니다. 하루 5번 밖에 제출 기회가 없는데 초반에 이론 공부를 하다가 기회를 많이 날린 것이 아쉬웠습니다. 제출 기회가 부족하여 여러 실험을 하지 못하였습니다. 이론 공부와 컴피티션을 동시에 진행하여, 충분히 기회를 사용할 필요가 있다고 생각했습니다. 후반부에는, 용량 문제와 디버깅 이슈로 인해, 모델 앙상블을 적용할 시간이 부족했습니다. 이 또한 미리 구현을 해놓고 프로그램이 문제 없이 돌아가는지 확인 해 두어, 제한 시간내에 원하는 실험을 해 볼 수 있도록 해야겠다고 느꼈습니다.
    4. 이번 Stage 에서는 보다 적극적인 정보공유로, 다양한 지식을 나눌 수 있었습니다. 정보 공유를 통해, 여러가지 실험에 대한 예측을 할 수 있었고, 시간을 줄일 수 있었습니다. 하지만, 직접 구현해낸 아이디어가 부족하여, 팀원들에게 많은 도움을 주지는 못하였습니다. 적극적으로 아이디어를 내고, 실험 결과를 공유하여, 팀원들 모두의 성장이 될 수 있도록 노력 해야겠다고 생각했습니다.

<마주한 한계와 도전숙제>

* 아쉬웠던 점들
  1. 나만의 개성있는 전략을 고안을 하였으나, 구현력의 한계로 결과를 내지 못한 점.
  2. 다양한 시도를 해 보지 못한 점.
  3. 처음부터 끝까지 baseline 코드를 작성해보지 못한 것
  4. 팀원들에게 다양한 정보를 제공해주지 못한 점
* 한계/교훈을 바탕으로 다음 스테이지에서 새롭게 시도해볼 것
  1. 다양한 논문을 읽고, 구현해 보는 연습을 할 것
  2. 다양한 방식으로 실험을 해보고, 컴피티션 제출 기회를 모두 사용할 것
  3. 토론 게시판과 피어 세션을 활용하여 정보 공유를 적극적으로 할 것
  4. 스스로 전체적인 baseline 코드를 작성해 볼 것
  5. 한가지 문제에 대해 깊게 파고들어 해결해 볼 것