Open-Domain Question Answering Project

8조 MIML

[1] 프로젝트 주제

지문이 주어지지 않은 질문에 대해 사전에 구축된 Knowledge Resource 에서 질문에 답할 수 있는 문서를 찾아 해당 문서에서 질문에 대한 답변을 추출하는 문제

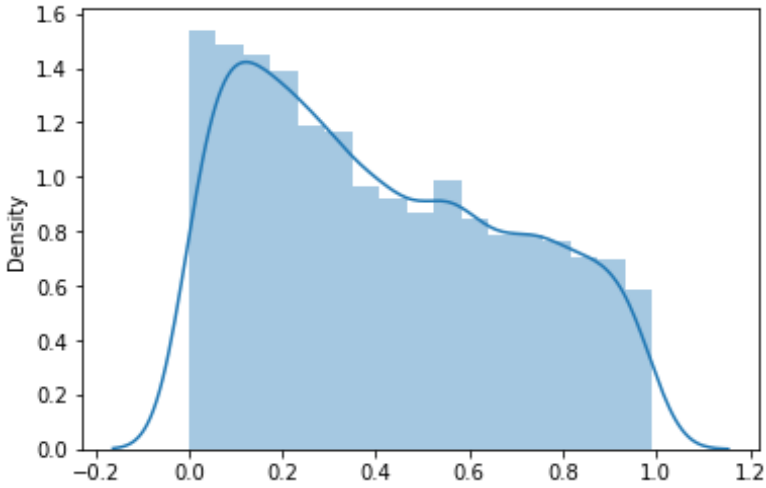
[2] 프로젝트 팀 구성 및 역할

조원	역할
임동진	Pretrained Model 및 Model Classifier 변경, Freezing, Data 변경
정재윤	Data 정제, Data Augmentation, Hyperparameter Tuning
조설아	Voting, Dense Retriever, Hybrid Retriever
허치영	KorQuAD 데이터 추가, T5 Modeling
이보림	Elastic Search, BM25, Hyperparameter Tuning, K-Fold Cross Validation, 〈〈Add Title〉〉, Preprocessing

[3] 프로젝트 수행 절차 및 방법

1. EDA 및 중복 데이터 확인

제공된 train dataset 은 3952 개의 매우 적은 양으로, augmentation 의 중요도가 높아졌다고 판단했습니다. 전체 3952 개의 context 중, 612 개의 중복 context 가 존재하였고, 한 context 에서 여러 개의 question 이 존재하는 것을 확인했습니다. 이번 Task 의 핵심은 '정답'에 있다고 판단하여 'answers'열을 중심으로 분석했고, 상당수의 정답 위치가 앞쪽에 쏠려 있음을 확인했습니다.



〈 Context 내 answer 의 위치 〉

2. Data Augmentation

1) Sentence Shuffling

정답의 분포가 context 앞부분에 치우쳐져 있는 것을 확인했습니다. Context 내 문장 순서를 섞더라도 문맥 파악과 정답 예측에는 큰 영향을 주지 않을 것이라 생각하여 context 내 문장 순서를 섞은 것을 이용해 실험을 진행했습니다.

Shuffled context 만을 이용한 경우 EM score 가 약 1.2 점 상승했고, 원래의 데이터와 함께 사용한 경우 오히려 성능이 하락했습니다.

단순 데이터 증강보다는 다양한 context 를 이용하는 것이 ODQA task 에서 더 중요하다는 것을 깨달을 수 있었습니다.

2) KorQuAD

주어진 train dataset 의 크기가 약 4000 개가량으로 학습에 사용하기에는 데이터가 너무 적다고 판단하였고, 외부 데이터셋을 사용하고자 했습니다. 한국어 Question Answering 데이터셋 중 가장 유명한 KorQuAD 를 사용하기로 했고, 그 중에서도 v1.0 을 이용했습니다.

KorQuAD v1.0 의 train dataset 은 총 60,407 개의 데이터를 가지며, 함께 학습에 사용할 경우 약 15 배의 추가 데이터셋을 사용할 수 있었습니다.

중복 context 를 확인한 결과, 총 9606 개의 context 를 사용하여 한 context 에서 여러 질문을 만들어낸 것으로 확인했습니다. 다만 context 내 다른 문장에서 정답을 가져오기 때문에 60,000 개의 데이터를 그대로 사용하기로 했습니다. 그 결과 약 2 점가량의 EM 성능 증가를 불러왔습니다.

```
KorQuAD: Dataset({
    features: ['id', 'title', 'context', 'question', 'answers'],
    num_rows: 60407
})
Datasets: Dataset({
    features: ['__index_level_0__', 'answers', 'context', 'document_id', 'id', 'question', 'title'],
    num_rows: 3952
})
```

KorQuADv1.0 에는 Model 학습 시 사용되는 features (id, title, context, question, answers)는 동일하게 담고 있어서 기존 데이터셋의 "__index_level_0__", "document_id"만 제거하면 바로 사용 가능했습니다.

3) Question Generation

자연어 프레임워크 중 하나인 'pororo'를 활용하여 context 중 적절한 단어를 랜덤으로 선택해 answer 로 삼고, question 을 생성.시간이 부족해 제출해보지는 못했지만, 생성해낸 answer 가 모두 5 자 이내의 단어였기에 성능향상을 기대하기는 어려웠을 것으로 추정

3. Preprocessing

1) Title 추가

Retriever Embedding 시에 추가 정보를 제공하고자 title 과 context 사이에 ' @ ' 구분점을 주고 이어 붙였습니다. 성능이 2 점가량 하락하여 사용하지 않았습니다.

2) 질문 형식 변경

"자동차를 발명한 사람은?"이라는 질문을 "자동차를 발명한 사람은 누구인가?"와 같은 형태로 질문을 보강시킬 경우 성능 향상을 불러왔다는 연구 결과에 따라 질문의 형태를 변경해보기로 했습니다.

질문을 영어로 번역한 뒤, 의문대명사를 활용하여 질문의 형식을 변경하여 사용했고, 그 결과 private leaderboard 상 1.4 점의 EM score 상승을 불러왔습니다.

3) 괄호 제거

Validation data 에서 틀린 결과를 확인했을 때, 괄호가 포함된 예측이 많이 포함된 것을 알게 되었습니다. ("트랜스포머" : 정답, "트랜스포머(Transformer)" : 오답)

모델 평가 방식이 EM 이였기 때문에, 괄호로 감싸진 부분을 답변에서 제거하고자 하였고, 그 결과 public leaderboard 에서는 성능이 약 1 점 가량 하락하였으나 private leader board 에서는 2.2 점이 상승하였습니다. 일부 경우에 대해서는 점수 상승의 효과가 있다는 것을 알 수 있었습니다.

4. Hyper parameter tuning

1) Top K Context (Retriever)

Retriever 에서 선택할 context 를 상위 10(base), 20, 25, 30 순으로 성능을 비교했습니다. 그 결과 public leaderboard 기준 20 〉 25 〉 30 순, private leaderboard 기준 30 〉 25 〉 20 순으로 성능이 좋았습니다.

2) Epoch

3, 4, 8 epoch 에 대해 실험을 진행했고, 대부분 3 과 4 사이에서 큰 차이는 없었으나 주로 3 epoch 에서 가장 좋은 성능을 나타냈습니다.

3) Weight Decay

0, 0.01, 0.02 세 종류에 대해 실험을 진행했고, weight decay 가 없는 경우 가장 좋은 성능을 나타냈습니다.

4) Batch Size

최근 연구된 모델에서는 batch size 가 증가할수록 더 좋은 성능을 나타내기에, batch size 를 키우고자 했으나 리소스 부족으로 인해 16 이상으로 키우지 못했습니다. 그래서 gradient accumulation 으로 batch size 를 키우는 효과를 나타내도록 했습니다. Gradient accumulation 을 4 로 설정하여 batch size 를 64 와 비슷하게 설정했을 때 private leaderboard 기준 EM score 5 점가량의 성능 향상이 있었습니다.

5. 모델 구현 및 모델 별 성능 비교

1) Pre-trained Language Model

KLUE/RoBERTa-Large, XLM-RoBERTa-Large, KE-T5 모델을 사용하고자 했습니다. 그 결과 T5 는 구현에 실패하였고, XLM-RoBERTa-Large 는 KLUE/RoBERTa-Large 에 비해 성능이 떨어져 **KLUE/RoBERTa-Large** 을 최종 선택했습니다.

2) Modeling T5

T5 는 huggingface 에서 T5ForConditionalGeneration 으로 QA 모델을 만들어야 하는 상황이었습니다. 그래서 직접 Question Answering 모델을 제작하고자 했고, baseline 코드 기반으로 T5 QA 모델을 제작하고자 하였으나 기간 내에 huggingface 라이브러리와 베이스라인 코드를 이해 하지 못해 구현에 실패했습니다.

3) Classifier 변경

Method	실험 이유	증가 폭 (EM; Private)	평가
Train only Classifier	Classifier 만 학습시킴으로써 Task 에 더 Specific 해지지 않을까 생각하여 수행	제출 X	Pretrained LM 을 downstream task 에 적용하기 위해서는 classifier 뿐만 아니라 모델 전체를 학습해야 한다.
CNN Layer	CNN 과 Transformer 를 조합하면 좋은 점수가 나온다는 말을 들은 것 같아 시도	-1.4	CNN 으로 수행하였을 경우, 과거 Data 가 잘 보존되지 않는 것 같다고 생각됨
BiLSTM	Context 와 질문을 동시에 고려해야하는 Task 이므로 이전 문장에 대한 정보가 어느정도 존재하는 BiLSTM 이 좋은 성능을 낼 수도 있을 것이라 생각	**+1.4**	BiLSTM 이 과거 Text 에 대한 정보를 어느 정도 담고 있어 매우 과거 정보였던 질문에 대한 데이터도 담아 학습이 진행되어 좋아진 것 같음

4. Retriever

1) Sparse Retriever (Base: TF-IDF Embedding)
   a) BM25
   문서의 길이를 반영한 BM25 중에서도 BM25, BM25+ 2 가지를 실험하였습니다. 가장 먼저 실험한 BM25 는 오히려 EM 이 10 점가량 점수가 하락하였습니다. 두 번째로 실험한 BM25+는 BM25 를 포함한 여러 랭킹 함수 들에서 긴 문장이 불리한 현상을 해결한 모델로 이를 이용하니 10 점가량 성능향상이 있었습니다.
   b) Elastic Search
   Elastic Search 검색엔진을 bm25 similarity 옵션을 주어 사용하였습니다. Base 에 비해 EM score 5 점이 상승했으나, BM25 보다는 public leaderboard 상 성능 향상이 좋지 않아 사용하지 않았습니다. 하지만 private leaderboard 에서는 성능이 오히려 향상된 것을 확인했습니다.
   파라미터 셋팅이나 elastic search 기능이 많았던 거에 비하여 많이 활용을 하지 못한 것 같아서 아쉬웠습니다.
2) Dense Retriever
   a) RoBERTa Model 의 Encoder 파트를 사용하도록 클래스를 정의하였습니다.
   b) 각 인코더를 학습시킬 때 진행하는 negative sampling 시, bm25 retriever 을 통해 얻은 유사도 score 및 index 리스트를 활용하여, 정답과 매우 유사하지만 오답인 것들을 sampling 하여 오답을 거르는 기능을 고도화하였습니다.
   c) 하지만 코드 간의 상호 작동이 원활하지 못해 활용이 미흡하였습니다
3) Hybrid Retriever
   a) Dense Retriever 로 구한 유사도 값에 Sparse Retriever 로 구한 유사도 값을 단순 합산하여 상위 N 개의 passage 리스트를 반환하는 방식을 취하였습니다.
   b) 위의 Dense Retriever 에서 발생한 문제로 인해 활용이 어려웠습니다.

5. Voting

Test dataset 성능이 좋은 순으로 prediction 파일을 정렬한 후, 각 data 의 결과를 hard voting 했습니다. 동일한 횟수로 정답이 예측될 경우 정렬 시 더 앞에 있는 파일의 예측 값을 정답으로 선택했습니다. Private leaderboard 결과는 더 많은 predictions 를 voting 할수록 더 좋은 성능을 나타냈습니다.

6. Ensemble

1) Fold 별 ensemble
   주어진 train dataset 의 문서길이를 고려하여 각기 다른 5 개의 train set 으로 나누고, 각각의 경우에 대해 학습한 모델들을 hard voting 을 통해 최종 예측 결과를 만들었습니다. 그 결과 EM 이 5 점 가량 성능 향상을 보였습니다.
   모든 데이터셋의 활용보다 다양한 데이터셋을 활용한 voting 방식이 더 효과적인 성능 향상을 불러옴을 알 수 있었습니다.

2) 성능 검증된 모델 내에서의 ensemble
   각 모델들의 성능 차이가 크지 않아 상위 몇 개 모델들의 ensemble 은 성능의 향상을 크게 가져오지 못하리라 예상하여 리더 보드 기준 75 점 이상의 성능을 나타내는 모델 중 각기 다른 기법을 적용한 7 개의 모델에 대해서 soft voting 을 진행했고, 개별 모델보다 1 점 가량 성능 향상을 보였습니다.

[4] 자체 평가 의견

전반적으로 아쉬움이 많이 남았던 대회였다고 생각합니다. Baseline code, Elastic search 등 코드와 라이브러리에 대해 깊이 이해하지 못해 더 다양한 실험을 진행하지 못했습니다. 다만 이전 대회의 경험을 토대로 곧바로 협업에 착수하여 협업 과정이 이전보다 원활하게 진행된 것은 발전한 점이라고 생각합니다.

개인회고: 임동진

나의 목표

먼저 Pretrained Model 변경보다는 Classifier 변경에 많은 힘을 쏟았다. 또한 저번 대회에서는 Classifier 를 변경시키는 것보다는 데이터 자체를 변환시키는게 더 좋았기 때문에 이를 고려하여 데이터 변경에 조금 더 힘을 써 성능 향상을 시도해보았다.

무엇을 어떻게 했는가?

이번 대회는 사실 BaseLine Code 부터 이해하기 매우 난해했던 것 같았다. 이전 대회에서 협업이 제대로 되지 않은 가장 큰 이유가 Hyperparameter 변경을 직접 입력하다보니 충돌이 나는 것으로 생각하여 최대한 모듈화를 진행하며 yaml 파일을 활용하여 Hyperparameter 를 설정하기로 하였다.

1 주차 때는 강의를 듣고 BaseLine Code 를 이해하는 데에만 바빴다. 이번 ODQA 라는 Task 자체가 매우 생소했기 때문에, 강의를 무조건 다 들어야 이번 대회를 참가할 수 있는 정도라 강의를 열심히 듣는데에 집중했다. 또한, 오피스아워와 나 나름대로 Huggingface 등의 Source Code 를 뜯어보며 BaseLine Code 를 이해하는데 큰 노력을 했다.

2 주차 때는 Classifier 를 변경시키는데 많은 노력을 기울였다. 가장 먼저 활용한 것이 Linear 를 여러 개 쌓아보는 것이였고, 다음에는 BiLSTM, CNN Layer 도 활용해보았다. BiLSTM 은 사용 방법이 조금 난해하여 처음엔 에러가 많이 떴지만 StackOverflow 와 Pytorch API, 활용 예시 등을 보며 사용법을 익혀 활용하였다.

3 주차 때는 데이터 변경에 많은 힘을 쏟았다. 이번 대회는 Title, Context, Question 을 수정할 수 있었다고 생각했다. 내 생각에는 Title 은 이번 대회에서 거의 못쓸 것 같고 Context 는 수정하는 방법이 떠오르지 않았다. 따라서, Question 부분을 뜯어보며 질문의 형식도 바꿔보고, 채점 방식을 알고 Answer 의 ()를 지워보는 등의 시도를 해보았다.

아쉬운 점 및 다음 대회 때 시도할 점

먼저, 저번 대회에서 느꼈던 점을 이번 대회에서 그대로 활용했다는 점에서는 나에게 큰 칭찬을 하고 싶다. 시간이 오래 걸리긴 했지만 BaseLine Code 를 잘 이해했으며, 협업을 위해 yaml 파일도 팀원 다 같이 만드는 등 훨씬 부드러운 협업이 된 것 같다. 또한 Source Code 도 많이 읽어보며 매우 힘들었지만 저번 대회보다 큰 에러 없이 많은 실험을 할 수 있었다.

아쉬운 점은 Task 의 난이도에 비해 시간이 부족했던 것 같다. 사실 Curriculum Learning 코드를 짜보고 싶었는데 데이터 변경 및 Korquad 를 활용하는, 내가 무조건 점수가 좋아질 것 같다고 생각하는 방법에서 점수가 오르지 않자 그 부분에 매달리다 마지막날 부랴부랴 Curriculum Learning 을 하다보니 분명 효과가 있을 거 같은 방법인데 제대로 실험을 하지 못했다는 점이 아쉬웠다.

두 번째로 2 주차 때 아파서 주말을 날렸던 점이다. 아팠더라도 Hyperparameter 정도는 변경시켜보면서 실험은 간간히 돌릴 수 있었을 것 같은데 그냥 조금 아프니까 쉬어야지! 했던 것이 나중에 시간이 부족했던 점과 합쳐져 너무 아쉬웠던 것 같다.

세 번째로 내 의견에 확신이 없었던 것이다. 사실 BiLSTM 및 Question 을 변경시키면서 점수가 오르지 않아 말이 안된다고 생각하면서도 “어쩔 수 없지..”라고 생각하며 그 방법을 포기하였다. 그런데 Private 점수를 보니 내가 활용했던 방법들이 모두 점수를 높이는데 큰 영향을 주었고, Korquad 도 내가 한 방식이 아닌 허치영 팀원이 한 방식으로 Concat 하니 점수 향상이 존재하였다. 제출 점수에 큰 의미를 두지 않고 내가 점수가 오를 것이라 생각한 이유가 타당 하다면 자신감을 가지고 계속해서 실험을 수행하면서 제출을 더 많이 했다면 좋았을 것 같다. 그렇다면 Private 점수는 더 좋아지지 않았을까라는 아쉬움이 든다.

다음 대회 때는 나의 선택에 조금 더 자신감을 가지는 게 좋을 것 같다. 또한, Unsupervised Model 을 활용할 때는 모델 변경도 중요하지만 EDA 를 통한 데이터 변경을 활용할 때 더 좋은 결과를 내는 것 같아, 품질 좋은 데이터를 만드는 방법에 대해 조금 더 많이 생각해봐야 할 것 같다고 느꼈다. 개인적으로는 이전 대회와는 달리 내가 시도해보려고 했던 대부분을 수행해보았고, Private 기준으로 내가 생각했던 방법론과 결과가 어느정도 일치했던 것 같아 만족했던 대회였던 것 같다.

개인회고: 정재윤

- **이번 프로젝트에서 나의 목표는 무엇이었는가?**

1. ODQA 의 전체 흐름을 이해하고 그에 걸맞는 기법 적용해보기
2. 협업에 Git 을 더 적극적으로 활용하기

- **나는 내 학습목표를 달성하기 위해 무엇을 어떻게 했는가?**

1. 이전과 다른 데이터셋 형태에 익숙해지기
2. 현재 TASK 에 맞는 전처리기법 탐색

- **나는 어떤 방식으로 팀에 기여했는가?**

1. 다양한 hyperparameter 변형을 통한 실험결과를 토대로 최적의 hyperparamter 를 탐색
2. 여러 augmentaion 기법들과 모델학습에 알맞는 데이터 변형을 활용하여 모델을 학습시킬 데이터셋 제작

- **내가 한 행동의 결과로 어떤 지점을 달성하고, 어떠한 깨달음을 얻었는가?**

1. 언어학적으로 접근했을 때 성능이 괜찮았던 이전의 대회경험을 참고해서 데이터셋을 제작했고 미미하지만 성능이 소폭 개선되었다.
2. 다양한 형태의 데이터셋을 적용했을 때 robust 한 모델이 될 것이라고 예상했고, 여러 validation set 을 구축하여 실험해보았지만, 실험결과는 썩 좋지는 않았다. 질문의 형태던 context 에 noise 를 주는 등 여러가지 기법을 적용해 데이터셋을 변형해보았지만 기계가 이해하기에는 그 기준이 명확하지 못했다고 생각했고, 이부분에 대해서 공부하고 정리해보아야겠다는 생각이 들었다.

- **전과 비교해서, 내가 새롭게 시도한 변화는 무엇이고, 어떤 효과가 있었는가?**

1. hyperparameter tuning 을 할 때, yaml 파일을 활용하면서 실험하기도 편했고, 결과를 공유 및 관리하기 쉬워졌다는 점에서 불필요한 노력을 줄이고 효율적으로 협업을 한 것 같아서 만족했다.
2. EDA 를 시도할 때, 주어진 파일만을 가지고 분석하는 것이 아니라 이번 TASK 를 생각해보면서 차근차근 진행했고 시각화에 공을 들여 주어진 데이터의 형태,성질을 한눈에 파악하려고 노력했다. 처음에 주어진 데이터에 익숙하지 않아서 EDA 파일을 참고하며 진행했고 초반 대회 진행에 많은 도움이 되었다.

- **마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?**

1. 주어진 baseline 코드를 이해하고 idea 을 생각해내느라 실질적으로 대회에 참여한 시간이 상대적으로 적어 생각해낸 기법을 모두 적용해보지 못해 아쉬웠다.
2. 이전 대회와 비교해봤을 때, 적절한 augmentation 기법을 많이 떠올리지 못했고, 많은 시도를 해보지 못한 것이 아쉬움으로 남았다.

대회 후기

이전 과는 다른 dataset 을 경험해봄으로써 다양한 dataset 을 접해볼 수 있어서 좋았습니다. 이전대회 뿐만 아니라 언어학적 접근이 생각 이상으로 좋은 성능을 보여준다는 것을 깨닫게된 대회였습니다. 또한 대회난이도가 올라간만큼 팀원간의 활발한 의견 공유가 더 중요함에도 불구하고 잘 이뤄지지 않은것같아 많이 아쉬웠습니다. 그렇지만 팀원들과 다양한 TASK 를 경험하고 협업을 함으로서 앞으로 이전보다 더 나은 팀프로젝트를 할 수 있다는 자신감을 얻었습니다.

개인회고: 조설아

- 코드 리팩토링, 디버깅, 함수화 및 모듈화, TDD 의 중요성을 알게 된 대회였습니다.
    - retriever 파트를 맡았으나 제대로 작동되지 않는 등 미흡했습니다.
    - 많은 기능을 한 번에 구현하려고 하다보니 어느 순간 코드가 꼬여 수습하지 못했습니다.
    - 기능을 잘게 나누어 모듈화하는 것의 필요성을 깨달았습니다.
    - 하나씩 차근차근 구현하고, 틈틈이 디버깅하며 유닛 테스트를 진행해보는 행위가 필수적이라는 것을 알게 되었습니다.
    - 가독성 있는 코드와 유의미하고 직관적인 변수명 네이밍, 정의된 함수에 대한 주석 달기 등 보기 좋은 코드를 작성하는 것은 생각보다 훨씬 중요한 파트였습니다.
    - 재사용성을 고려하면서 코드를 작성하기 위해 적절한 라이브러리나 파일 형식을 취해야함을 깨달았습니다.
- 틈틈이 디버깅하고, 작은 기능이라도 함수화하고, 기능별 구분이 가능할 경우 모듈화하면서 각각 유닛테스트를 통해 잘 작동하는지, 어떻게 작동하는지 인지한 후 코드 리팩토링을 통해 코드의 가독성과 재사용성을 높일 것입니다.

개인회고: 허치영

Baseline code, huggingface 라이브러리 이해의 필요성

이번 대회에서 T5 모델을 이용해서 실험을 진행하고자 했습니다. Huggingface 라이브러리에서 따로 제공해주는 T5 의 Question Answering 모델이 없기에 직접 만들어야 하는 상황이었습니다. 관련 문서와 baseline code 를 계속해서 공부했으나 생소한 코드의 형태와 완전히 익숙하지 않았던 huggingface 라이브러리였기에 기간내에 T5 QA 모델을 구현하지 못했습니다. Huggingface 라이브러리에서 제공하는 함수와 클래스들에 대해서 생소했던 것이 주된 원인이라고 생각했습니다. 각 함수의 입출력과 사용되는 파라미터에 대해서 깊게 이해하지 못하다 보니 어디서 오류가 발생했는지 감조차 못 잡는 상황이 잦았습니다. 이후 product serving 에서도 사용할 huggingface 라이브러리이니 huggingface 에 대한 좀 더 심도 깊은 탐구가 필요하리라 생각했습니다.

품질 좋은 데이터셋의 중요성

이번 대회에서 제공되는 데이터셋의 크기는 4000 개 미만으로 적은 양의 데이터셋이였습니다. 다만 외부 데이터셋을 허용했기에 KorQuAD 데이터셋을 추가적으로 활용했고, 그로 인해 성능 개선을 얻어낼 수 있었습니다. KorQuAD 데이터셋에서 중복 context 를 제거한 후 사용하며 다른 추가 데이터셋을 활용할 수 있었다면 더 좋은 성능을 얻어낼 수 있지 않았을까 하는 아쉬움이 남았습니다. 좋은 성능의 딥러닝 모델을 위해서는 품질 좋고 많은 양의 데이터셋이 필수적이다는 사실을 알게 되었습니다. Product serving 기간 동안 데이터 품질 관리에 많은 신경을 기울이겠다는 다짐을 하게 되었습니다.

멘탈 케어의 필요성

이번 대회에 크게 번아웃이 와서 제대로 대회에 참여하지 못했습니다. 이전의 2 개 대회를 끝낸 후 제대로 휴식을 취하지 못하고 계속해서 공부했던 것이 원인이었던 것 같습니다. 공부에 많은 시간을 쏟는 것도 좋으나 하나의 대회 혹은 프로젝트가 끝난 후에는 충분한 휴식이 동반되어야 꾸준히 할 수 있다는 것을 느낄 수 있었습니다.

개인회고: 이보림

이번 프로젝트 목표

- ODQA 과정을 제대로 이해하고 실험에 적용 전에 이론을 이해하고 결과를 예상하며 실험해보기

팀에서의 내 역할 및 기여

- EDA 및 Retriever 위주의 실험
- 최종 앙상블 제외 시 팀 내 최고 성능 모델 제출

전과 비교해서 새롭게 시도한 변화 *모든 성능 비교는 EM 기준

- yaml 파일로 각각 다른 config 파일로 관리
- 새로운 모델이나 함수 적용 시 논문이나 document 읽고 이해 후 적용
- 무조건 성능을 높이기 보다는 프로세스 이해에 초점을 맞춤

한계 및 아쉬웠던 점

- 평가 방법이 Exact Match 여서 더욱 결과 예측이 힘들었다. 그래서인지 Public Score 와 Private Score 차이가 커서 실험을 제대로 진행하지 못했다.
- 위의 이유와 더불어 여러 실험들을 할 수록 Public score 성능이 떨어져서 결국 최종 제출 모델은 가설 적용을 거의 하지 않은 모델이었다는 점이 아쉬웠다. Private Score 에서 오를 거라는 것을 미리 예상했다면 더 많은 실험을 진행하고 적용해보았을 것 같다.
- Elastic Search 를 제대로 활용해보고 싶었는데 기능이나 파라미터들이 너무 많고 이를 친절히 설명한 문서들도 적어서 하나 하나씩 제대로 알기 힘들었다.
- 제출 시 성능이 잘 나왔던 모델에 대한 셋팅을 잘 못 관리하여 다시 구현하기 위한 시간이 많이 소요되었던 점이 아쉬웠다.

대회를 통해 알게 된 점

- ODQA 의 전반적인 프로세스와 관련 기술들을 알 수 있었다.
- Elastic Search 사용법이나 config 파일을 통한 Hugging face trainer 관리 등을 알게되었다.
- 저번 대회와 달리 시작 전, 깃허브 코드를 제대로 정리하고 시작하였다. 그래서 컨플릭트가 거의 없이 대회를 진행할 수 있어서 협업 시 시작 전 rule 을 정하는 것과 셋팅의 중요성을 직접 느낄 수 있었다.
- 평가 방식에 대한 이해가 중요하며 평가 방식에 따라 실험할 때의 방향이나 Public Socre 를 얼마나 반영할 지 등 많은 부분이 달라져야 한다는 것을 알게되었다.
- 성능이 가장 높았던 모델에 대한 셋팅 관리를 잘못했던 경험을 통하여 모델 버전 관리에 대한 중요성을 많이 느낄 수 있었다.