개인 회고 : 김소연

모델 개선과 학습 목표 달성을 위해서 시도해본 것, 그에 따른 결과들 :

☺ **어떻게 train/validation 을 나눌 것인가. Validation set 중복 문장의 영향 확인**

데이터 EDA를 통해 중복 문장 존재하는 것을 확인했습니다. 이전 대회와 마찬가지로 임의의 데이터셋을 training/validation으로 나눌 경우 훈련에 사용된 문장이 validation에 들어갈 수 있고, 그럴 경우 중복이 발생할 수 있다고 생각했습니다. 그렇기 때문에 저는 `sklearn`의 `train_test_split` 을 이용한 임의의 분리 방식을 베이스라인(split-basic)으로 두고 중복이 있는 문장도 stratify, 중복이 있는 문장은 대표 문장 기준으로 stratify로 나누어서 train/validation에 중복 문장 케이스가 들어갈 수는 있으나, 동일한 문장이 train/validation에 흩어져서 분배 되는걸 막도록 설계(split-dup)했습니다. 그리고 비슷한 철학으로 구현한 팀원의 방식(split-team)으로 함께 성능평가를 해보았을 때 성능 이득의 경향성이 상이했습니다. 추가 적용을 고민하기 전 팀원이 앙상블 구현을 완료한 상태라 이후 실험에서는 팀원이 구현한 방식으로 실험을 진행했습니다. 하지만 본 실험과 이후의 실험 과정에서도 크게 놓친 부분은 validation set을 고정시키지 않았다는 것입니다. 즉, 어떻게 구현하느냐에 따라 validation set이 달라지기 때문에 구현한 방식이 정말 성능 향상을 이뤘다고 판단하기에 부족했습니다. 또한, 달라지는 환경세팅에 따라 어느 체크포인트에 제출하느냐도 성능을 크게 좌우하는 것을 실험 중후반에 깨닫게 되어 본 검증 세팅의 성능 검증이 엄밀하지 못했던 것 같습니다.

BERT-base(F1/AUPRC)		Roberta-Large F1/AUPRC)		
Split-basic	**Split-dup**	**Split-basic**	**Split-dup**	**Split-team**
64.4653/68.7725	**65.8458/68.8073**	**67.6319/70.3416**	66.7867&70.0707	66.5051&67.1578

Roberta-large		Label smoothing+Bi-LSTM ver2+ Roberta large	
ckpt @ 1700	ckpt@ 4500	Ckpt @ 2400	ckpt@ 4800
64.2766/66.8118	**69.2046/69.7165**	68.3352/69.5783	**70.1896/66.4824**

☺ **Pretrained 모델 종류와 모델 구조에 따른 성능 추이**

Pretrained model 학습을 위해 사용할 수 있는 옵션은 bert-multilingual과 아예 한국어 데이터셋으로 학습된 모델이었습니다. 하지만 bert-multilingual 경우 전체 데이터 사이즈는 크나, 굉장히 많은 언어 대비 한국어 비중이 크지 않을 것이기에(그러나 정확히 한국어 비중이 얼마인지 찾질 못했습니다) 실험하지 않고 본 task의 데이터셋과 유사한 소스로(wiki, namuwiki) 학습된 KoElectra, klue/BERT, klue/Roberta 로 실험을 진행했습니다. 실험 결과는 Roberta-large가 가장 높게 나왔는데 KoElectra보다 좀더 구체적으로 klue dataset에 학습된 모델이며, 일반적으로 BERT가 Roberta보다 높게 나온 경향성을 따른 것이라 판단했습니다.

[CLS] 토큰만을 이용해 classifier에 넘기는 기존의 구조에는 fully connected layer hidden size를 2배, 3배로 늘리거나 layer를 좀 더 쌓는 방식으로 변화를 주어서 실험해보았으며, 전체 토큰 임베딩을 함께 사용해 bi-LSTM 구조를 이용하는 방식도 적용했습니다. 끝으로 RBERT 코드를 참고해 entity 위치의 임베딩을 사용하는 방식도 적용해보았습니다. 결과적으로 fully connected의 hidden size를 늘리는 것이나 bi-LSTM을 적용하는 것이 성능 향상을 주었고 RBERT 경우 앞선 구조보다 2퍼센트 정도 낮은 성적을 냈습니다. 또한, 팀원들의 추가 실험들을 통해 entity typin을 넣을 경우 bi-LSTM보다 fc layer를 쌓은 구조를 사용하는 것이 더 성능을 높여주었는데, 이는 entity가 갖는 임베딩들을 lstm에 넣는 것보다 [CLS]토큰을 classifier에게 넘겨주는 것이 모델 학습을 좀더 깔끔하게 할 수 있도록 도와줬다고 생각합니다. RBERT의 앙상블값은 single roberta-large 성능과 비슷하게 나오는 수준이라 최종 모델 선택에서 제외하였습니다. (아래 테이블은 체크포인트 제출값이 후반부랑 달라 다소 점수는 낮게 표기)

KoElectra	**Bert-base**	**Roberta-large**	-
60.5327&54.5327	64.4653/68.7725	**67.6319/70.3416**	-
Averaging token embedding	**FC classifier(hidden size 2배)**	**B-iLSTM**	**Bi-LSTM ver2**
57.7162//68.0450	66.1042/71.0682	66.5228/71.9097	69.5625/71.8766

☺ **불균형한 데이터셋의 악영향을 잡아줄 수 있는 loss 함수를 허깅페이스 호환으로 적용**

Baseline 코드를 이용해 학습할 경우 loss 변경이 쉬우나 저는 baseline code를 짜서 학습한 경험이 몇 번 있었기에 복잡한 라이브러리를 수정하면서 커스터마이징하는 것에 익숙해지는 것이 목표였습니다. 따라서 Trainer의 method 진행 흐름을 파악, compute_loss 메소드 오버라이딩과 constructor 부분을 수정해 실험하고 싶은 손실함수를 적용했습니다.

기본적으로 class imbalance가 크기 때문에 focal loss, ldam loss를 적용해보았습니다. Trainer의 evaluate 메소드와 wandbCallback 클래스를 오버라이딩해 confusion matrix로 분석해본 결과 ldam loss는 valdiation에서 no_relation에서의 성능이 굉장히 떨어졌고 그 영향이 다른 클래스에도 미치는 경향성을 띄었습니다. Focal loss 는 class weight를 고려하면 loss 값이 너무 작게 측정되어 class weight distribution을 따로 주지 않은 것과 비교했을 때 leader board의 성능차이가 크게 나지 않았습니다. 마지막으로 class imbalance보다는 generalization에 비중을 두는 label smoothing을 적용했을 때 가장 큰 효과가 있어 이를 향후 학습에 적용하였습니다. Confusion matrix를 분석해보면 적은 개수의 데이터를 가진 클래스의 discriminative power가 떨어지긴하지만, 비슷한 성격의 클래스(e.g, place of residence, origin, place of birth / other family, parents, siblings)가 존재하고 그 사이에서 헷갈려하는 영향이 큰 것으로 보였습니다. 따라서 hard label에 대해서만 높은 확률로 맞추게 하는 것보다, 오답 클래스에 대해서도 일정 거리를 둘 수 있게 하는 label smoothing이 더 좋은 성능을 주었다고 생각합니다.

LDAM loss	Focal loss	Label smoothing(0.1)
56.6861/59.4016	66.9481/67.8623	**69.7276/67.2901**

☺ **Batch size, learning rate 튜닝**

일정 epoch 이후 빠르게 오버피팅 되는 것을 방지 하기 위해 팀원이 batch size를 키우고, learning rate를 조절하는 것을 제안했습니다. 허깅페이스, pytorch lightening의 fp16 사용, 메모리를 덜 먹는 Adafactor 사용으로 batch size(BS) 조절을 통한 regularization이 가능해졌고, 더 빠른 속도로 실험을 수행할 수 있었습니다. 특히 BS를 2배로 늘리고(32 → 64), learning rate 감소(5e-5 → 2e-5) 시 성능 향상 폭이 가장 높았는데, 각각 떼어내어 실험해본 결과 두가지 모두 비슷한 성능 향상 폭을 내는 것을 통해 역시나 learning rate가 진짜 중요하다는 것을 다시 한번 더 느끼게 됐습니다.(BS 조절도 Learning rate 조절로 볼 수 있다는 점에서) 그렇기 때문에 다른 팀원이 entity typing 테스트 시 높게 나와야 하는 성능에서 낮게 나온 이유에 대해 GPU에 가능하게 꽉 채워 BS를 늘려 학습해보는 것을 권해드렸고, 예상하는 결과 경향성을 얻을 수 있었습니다.

(아래는 stratified KFold 5, BS64로 고정, learning rate만 수정)

5e-5	2e-5	1e-5
74.0418/78.0355	**75.5070/76.1593**	75.2015/76.2392

☺ **Stratified KFold 와 KFold. 기본 데이터와 증강데이터 → 데이터 양을 변형해 학습에 사용**

원래 학습 데이터를 8:2로 잘라 StratifiedKFold(5)로 학습시켜보고, 9:1로 학습시키기 위해 임의로 분할하는 KFold(10)으로 학습시켜보았습니다. 실험 당시에는 전체 데이터 중 학습에 사용하는 양을 조절하려면 'k' 값 수정을 통해서만 가능하다고 생각했었고, 그럴 경우 minor 클래스가 갖는 양이 너무 작을 것 같아서 StratifiedKFold(10)대신 KFold(10)을 선택해서 실험했습니다. 그 결과 KFold(10)을 한 실험이 팀 내에 가장 높은 값을 찍었습니다.(마지막 날 제출용 csv 끼리의 soft voting 제외 제일 높은 점수). 다만, 이 결과가 학습에 사용되는 데이터가 조금 더 증강되어서인지, 아니면 5개의 모델의 앙상블 vs 10개 모델의 앙상블에 대한 이득인가에 대한 비교는 하지 못했습니다. 다른 아쉬운 점으로 저는 KFold의 베이스라인 생성 겸으로 다른 팀원들이 증강해둔 데이터셋을 사용하지 않고 기본으로 주어진 train.csv와 중복만 제거하는 간단한 전처리가 진행된 데이터셋을 사용했었습니다. 그러나 높은 결과값을 얻은 후 동일한 세팅으로 팀원들과 함께 증강 데이터 사용 및 다른 기법들을 적용했음에도 불구하고 근소한 차이로 계속 낮은 점수를 기록했습니다. 이 실험에 대한 엄밀한 분석이 필요한 부분도 있지만, 첫번째에 언급했듯 validation set이 고정되지 않았기 때문에 증강된 데이터를 사용하는 경우와 기본 데이터를 사용해 얻는 validation set이 다르기 때문에 f1 score, loss를 분석하는 것의 의미가 없었습니다.(훈련에서는 8~9퍼센트 차이나도 LB에서는 낮은게 더 높게 나옴)

얻은 점, 깨달은 점, 아쉬운 점 :

이번 대회는 코드 공유와 협업을 경험하는 것, 허깅페이스 라이브러리에 익숙해지는 것이 우선적인 목표였습니다. 그런 면에서 더 발산되기 전에 코드를 통합하고, 어떤 식의 실험들이 수행되었는지 기록, 높은 성능 환경과 동일한 셋팅으로 점진적으로 쌓아나가면서 실험하는 방식을 취했습니다. 이전 대회에서는 동일한 실험 세팅을 주더라도 성적 reproduction이 아예 안되었기 때문에 공통된 가설을 검증하는 실험이 어려웠었기에 그런 부분을 크게 개선한 협업 경험이었습니다. 또한, 허깅페이스의 Trainer, Tokenizer, custom model 파일들, 여러 integration callback 클래스들을 직접 까고 custom으로 변형하면서 파악한 과정들을 notebook에 공유하고 팀원들에게 알려주면서 이해도를 높일 수 있었습니다.

초반에 각자 진행 속도가 너무 빠른데 비해 공유가 잘 되지 않는 인상을 받아 노션에 자세하게 각자 진행상황을 공유할 것을 건의했고 처음보다는 개선된 부분이 있었다 생각하지만, 여전히 어떻게 각자 수행한 실험을 필요한 부분만 집중해 효과적으로 전달할 수 있을지는 고민 해봐야 할 문제인 것 같습니다. 또한, 몇 번이나 언급한 validation set을 고정하지 않아서 모델의 엄밀한 평가가 이루어지지 못한 점을 비롯해 오히려 너무 일찍 앙상블을 시도해서 단일 모델로서 여러가지 variant들에 대한 성능 효과 검증이 많이 부족했다고 생각합니다. 앙상블이 기본적으로 낀 코드로 실험을 돌리고 공유하다보니 한번 성능 결과를 얻기까지 훈련 시간이 오래 걸리고 그 결과 팀 전체적으로 다양한 아이디어 수용에 보수적이게 되지 않았을까란 생각이 들었습니다.

구현 측면에서는 허깅페이스의 전체적인 틀과 모델 customize를 위해 어떤 부분을 고쳐야하는지는 파악했으나 bert, roberta의 학습 로직 구현, 동작에 관여하는 세부 모듈의 input, output 처리에 대한 이해도까지 섭렵하지는 못한 것 같아 아쉽습니다. 또한 이전 기수에서 주로 Roberta-large가 높게 나왔고 실제 실험에서도 높게 나왔으니 사용은 했지만 다른 한국어 모델이 왜 잘 안 나오는지에 대한 이해가 충분하진 않았던 것 같습니다.(가령 어떤 성향의 데이터셋, 얼만큼의 데이터셋 사이즈, 모델 사이즈 비교에 대한 수치적인 엄밀한 비교 없이 그냥 대세를 따라간 기분입니다.) 끝으로 어떤 세팅이냐에 따라 추론에 사용하는 체크포인트의 성능 영향도가 꽤 컸는데, 특히 스페셜 세션에서 "그냥 전체 데이터에 학습하고 일정 체크포인트에 앙상블을 시키는 것이 오히려 좋을 수 있다"란 말까지 들으며. 어쨌던 협업을 하면서 어떻게 제출 체크포인트 시점을 정하는 것이 좋을지에 대한 협의도 중요할 것 같다고 생각했습니다.

본 프로젝트를 통해 다음 프로젝트에서 시도해볼 것

이번 대회에서 크게 성능 이득을 가져다 준 것은 batch size, learning rate 조절, 앙상블, 모델 구조 개선, 손실 함수였던 것 같습니다. 이런 부분은 다양한 하이퍼파라미터 튜닝 라이브러리를 통해 더욱 적합한 값을 찾을 수 있었을 것 같습니다. 또한, 각각 EDA, 데이터 증강, 모델링을 하면서 나오는 아이디어들을 아이디어 수준으로 끝내는 것이 아니라 각자 검증하고 싶은 가설로 구체화를 하고, 단일 모델에 대한 검증까지 끝내서 해당 가설이 왜 동작하고 동작하지 않았는지에 대한 충분한 토론이 이루어질 수 있도록 실험하면 좋을 것 같습니다.