



주제

- 사전에 주어진 knowledge resource로부터 주어진 질의의 답변을 추론하는 태스크
- 질문과 관련한 문서를 찾는 retriever 모델과, 문서로부터 질문에 대한 답변을 찾는 reader 모델의 two-stage로 구성

일정

타임라인



Reader에 해당하는 MRC 학습데이터는 다음과 같습니다.

- columns
 - `id`: 질문의 고유 id
 - `question`: 질문
 - `answers`: 답변에 대한 정보. 하나의 질문에 하나의 답변만 존재함
 - `answer_start`: 답변의 시작 위치
 - `text`: 답변의 텍스트
 - `context`: 답변이 포함된 문서
 - `title`: 문서의 제목
 - `document_id`: 문서의 고유 id

Corpus 데이터

질문과 관련된 문서를 찾아오는 Retrieval 과정에 사용된 데이터는 `wikipedia_documents.json` 파일로 약 57,000개의 unique한 문서로 구성되어 있습니다.

데이터 폴더 구조

Wrap Up Report

평가방법

Exact Match Score

- 모델이 예측한 text와 정답 text가 글자 단위로 완전히 똑같은 경우에만 1점, 한 글자라도 다르면 0점을 부여합니다.
- 이때, 띄어쓰기나 "."과 같은 문자는 제외하고 점수를 산정합니다.
- 각 sample이 부여받은 점수의 합 / 전체 sample의 개수로 Exact Match Score가 계산됩니다.

In 1870, Tesla moved to Karlovac, to attend school at the Higher Real Gymnasium, where he was profoundly influenced by a math teacher Martin Sekulić.32 The classes were held in German, as it was a school within the Austro-Hungarian Military Frontier. Tesla was able to perform integral calculus in his head, which prompted his teachers to believe that he was cheating. He finished a four-year term in three years, graduating in 1873.33

Why did Tesla go to Karlovac?
Ground Truth Answers: to attend school to attend school attend school at the Higher Real Gymnasium
Prediction: to attend school at the Higher Real Gymnasium

출처 : <https://stages.ai/competitions/242/overview/evaluation>

F1 Score

- 모델이 예측한 text와 정답 text의 overlap으로 F1 값을 계산합니다.
- 모델이 예측한 text와 정답 text가 완벽히 일치하지 않아도 0~1 사이의 부분점수를 받을 수 있습니다.

The definition of true positive (TP), true negative (TN), false positive (FP), false negative (FN)

	Tokens in Reference	Tokens Not in Reference
tokens in candidate	TP	FP
tokens not in candidate	FN	TN

$Precision = \frac{num(same_token)}{num(pred_tokens)}$

$Recall = \frac{num(same_token)}{num(groud_tokens)}$

$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$

출처 : <https://stages.ai/competitions/242/overview/evaluation>

구성 및 역할

	역할
문지혜	Soft Voting 구현, 외부 데이터(KorQuAD) 도입
박경택	DPR 구현
박지은	EDA, validation inference 및 사후 분석, BM25 구현, Curriculum Learning, Training with Retrieved Context
송인서	Hard Voting 구현
윤지환	baseline code refactoring, EDA, Elastic Retrieval 구현, 하이퍼파라미터 실험

협업

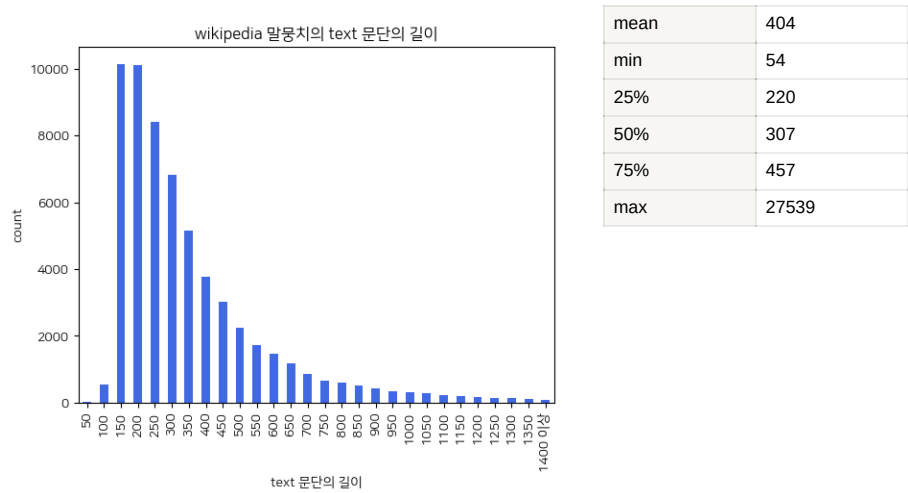
- GitHub 브랜치 전략 — Git flow
 - 대회 형식의 프로젝트에서는 GitHub flow를 사용하는 것이 더 적합하다고 판단했으나, 최종 프로젝트에서 사용하기 위한 시도 목적으로 Git flow 전략을 따랐습니다.
- 실험
 - 실험 자동화 및 관리로는 셸 스크립트(shell scripts) 활용하였으며, 실험 기록 플랫폼은 노션(Notion)을 활용하였습니다.

2. 프로젝트 과정

EDA

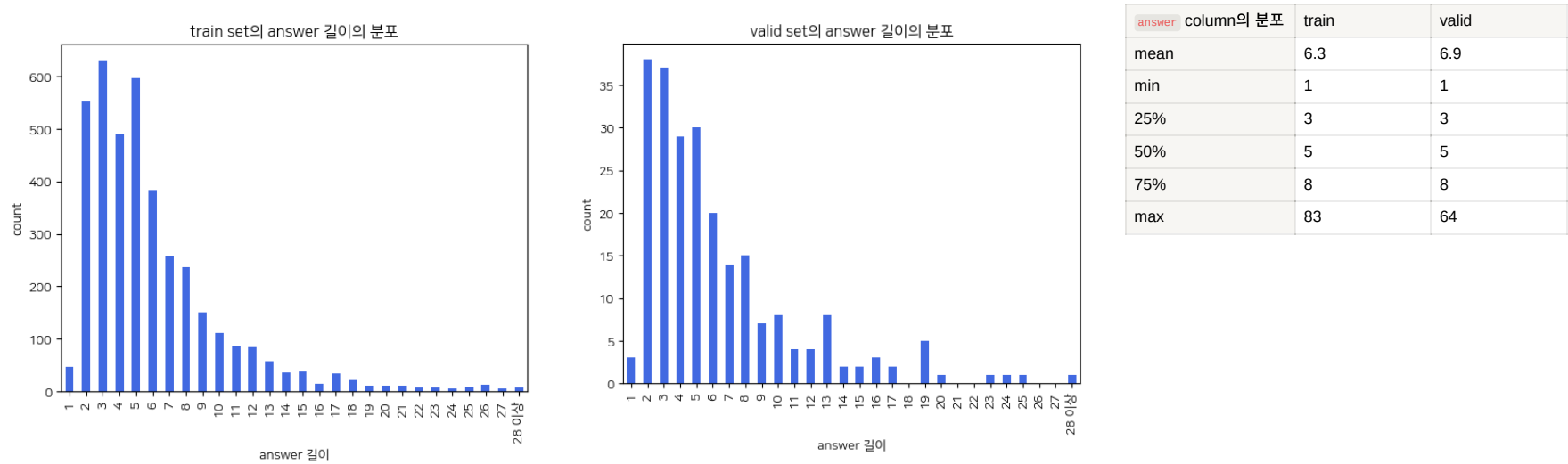
retrieval에 사용되는 corpus: wikipedia_documents.json

- corpus의 문단 길이의 분포



reader 학습에 사용되는 데이터: train_dataset , test_dataset

- answer column 길이의 분포
 - answer column의 길이 단위는, 다른 column들과 다르게 '토큰'이 아닌 '글자'를 단위로 사용하였습니다.
 - reader 모델이 context에서 answer의 index를 예측하는 방식으로 작동하기 때문에 string의 index를 세는 단위인 '글자'를 사용하였습니다.



- answer column 특수문자 및 외국어 포함 여부
 - 괄호 및 따옴표 등의 특수문자가 정답에 포함된 경우를 확인해볼 수 있었습니다.
 - ex. '초일기', <<인간의 이해: 개념의 집단적 사용 및 진화 (1972)>>, <성당과 시장> (The Cathedral and the Bazaar)

- **question** column 문장 길이의 분포



max	43	32
-----	----	----

- **context** column 문장 길이의 분포



max	1172	1151
-----	------	------

Frameworks Overview



- 이번 프로젝트에서는 Retriever-Reader 모델(왼쪽 그림의 첫 번째 및 오른쪽 그림) 구현에 집중하였습니다.

Retrieval

TF-IDF

- $$\circ \text{Score}(D, Q) = \sum_{\text{term} \in Q} \text{TF-IDF}(\text{term}, Q) \times \text{TF-IDF}(\text{term}, D)$$

$$IDF(d, t) = \log \left(\frac{N}{1 + DF(t)} \right)$$

- 전체 문서의 수 N 이 커질 수록 $IDF(d, t)$ 의 값이 기하급수적으로 커지는 것을 방지하기 위해 log 사용
- 진수의 분모가 0이 되는 것을 방지하기 위해 1을 더함

BM25

- BM25

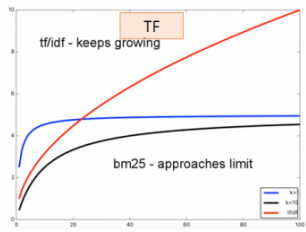
- $Score(D, Q) = \sum_{i=1}^n IDF(q_i) \cdot \frac{TF(q_i, D) \cdot (k_1 + 1)}{TF(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})}$

- TF-IDF의 단점을 보완한 알고리즘으로, TF-IDF 계열 information retrieval 알고리즘 중 SOTA로 알려져있습니다.

- 특징

1. 문서 길이 정규화: BM25는 문서 길이를 정규화하여 사용함으로써, 길이가 긴 문서가 상대적으로 더 높은 TF 점수를 얻는 것을 방지합니다.
 - TF-IDF에서는 TF가 단순히 특정 문서에서 단어의 등장 횟수를 나타내기 때문에, 문서의 길이가 길어질 수록 TF가 커진다는 특징을 가집니다.
 - BM25에서는 TF 부분의 분모에 문서 길이를 전체 문서의 평균 길이로 나눈 값($\frac{|D|}{avgdl}$)이 포함되어, 문서의 길이가 짧을 수록 더 높은 점수를 가지도록 합니다.

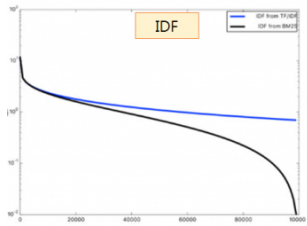
2. TF 정규화: TF-IDF와 달리, BM25는 TF를 정규화하여 사용함으로써 특정 단어가 문서 내에 반복적으로 많이 나타나더라도 그 단어의 중요성이 무한정 증가하지 않도록 합니다.



TF 정규화 그래프

3. IDF의 영향력 증가: BM25에서는 DF가 높아지면 IDF가 급격하게 0으로 수렴하여, 불용어가 중요도에 미치는 영향력이 적습니다.

- $idf(q_i) = \ln\left(\frac{N - df(q_i) + 0.5}{df(q_i) + 0.5}\right) + 1$



IDF 영향력 그래프

- 실험 결과

- 리더보드 제출 결과

	EM	F1
TF-IDF	56.2500	66.0400
BM25	63.7500	74.1600

- retriever의 정확도 (top_k = 30)

	accuracy
TF-IDF	80.0
BM25	95.42

- 전체 240개 데이터 중 TF-IDF는 48개를, BM25는 11개를 잘못 예측했습니다.
- BM25의 잘못된 예측 11개는 모두 TF-IDF의 잘못된 예측에 포함되는 값입니다.
 - 따라서 BM25와 TF-IDF의 관계는 trade-off나 보완 관계가 아닌 일방적인 성능 향상으로 볼 수 있습니다.

- context 길이 분포

context	column의 분포	valid	tfidf 오답	bm25 오답
mean		494.3	508.19	444.54
min		265	267	276
25%		339.25	365.25	355
50%		433.5	467.5	402
75%		597.25	645.75	502
max		1151	860	807

- TF-IDF는 오답 context의 길이가 상대적으로 긴 것을 확인할 수 있는데, BM25에서는 이러한 부분이 완화된 모습을 확인할 수 있습니다.

ElasticSearch

Elasticsearch는 Apache Lucene을 기반으로 구축된 검색 및 분석 엔진으로 텍스트, 숫자, 위치정보, 정형 및 비정형 데이터등 모든 유형의 데이터를 다룰 수 있습니다. Open Domain의 성격상 Retrieval의 대상이 되는 wikipedia 데이터의 크기가 크고 각 질문마다 적절한 passage를 찾아내는데 시간이 오래걸립니다.

Elasticsearch는 **Inverted index**(문서 내의 문자와 같은 내용물의 맵핑 정보를 색인해놓는 것. ex) 책 가장 뒤 단어 별 색인 페이지)를 사용한 분산처리를 통해 실시간으로 빠른 검색이 가능합니다. 베이스라인 대비 빠르고 효율적으로 top-k passsage를 가져오기 위해 시도하였습니다.

- 인덱스 생성 setting

- `nori_tokenizer` : 은전한닢 형태소 분석기 기반 한국어 tokenizer
- `shingle token filter` : 단어 단위의 한 묶음을 구성하는 filter
- `mixed decompound_mode` : 복합명사 분리하고 기존 형태를 보존
- `similarity score` : BM25

- 실험

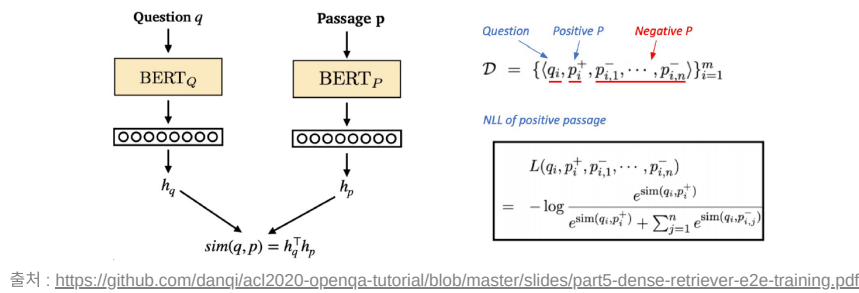
- 이번 대회는 Reader-Retrieval 모델 간 하이퍼파라미터 조합에 따라 성능이 천차만별로 달라져 실험내용이 변경될 때 마다 성능이 가장 좋은 조합을 다시 찾아야 했습니다. 때문에 실험 간 파라미터가 약간씩 다르게 적용되어 정확한 비교는 힘들었습니다.
- 기본 모델을 기준으로 bm25에서 elasticsearch로 변경한 경우 오히려 성능이 떨어졌습니다.
- 하지만 '##', *개행문자*($\ln, \backslash n$)와 같은 내용에 영향은 없지만 많은 비중을 차지하는 문자들을 **전처리한 데이터와 wikipedia data**로 reader model을 학습한 후 retrieval 을 실행한 결과 **EM 67.5, F1 77.12** 로 **EM 기준 4점** 정도의 성능향상을 보였습니다.
- Reader model과 Retrieval 간에 성능 면에서 맞는 조합을 잘 찾아야 한다는 점을 깨달았습니다.
- 다만 BM250kapi 라이브러리를 사용한 경우와 비교해 인덱싱 및 top-k passage를 찾는 속도에 차이가 거의 없었는데
 - wikipedia_document의 크기가 대용량 데이터가 아니여서 메모리 기반인 BM250kapi와 큰 차이가 없었거나
 - 작성한 코드에서 네트워크 호출이 필요한 search 메서드가 여러 쿼리에서 한 번에 실행되는 것이 아닌 각 쿼리를 개별적으로 실행하고 있기에 차이가 없지 않았을까 파악하고 있습니다. 추후 코드를 개선해볼 예정입니다.

Model(top-30)	EM	F1
baseline(tf-idf)	56.2500	66.0400
baseline + bm25	63.7500	74.1600
baseline + elasticsearch	61.6700	72.25
baseline + elasticsearch + data preprocess	67.5000	75.95

DPR (Dense Passage Retrieval)

- DPR 소개

- 참고할 지문(Passage)와 질의(Question, Query) 각각에 대해 dense embeddings를 산출하고, 두 embedding vector를 내적하여 유사도를 구합니다. 이때, Passage와 Question에 대한 encoder를 따로 정의하여 구성하는 Bi-encoder 방식을 먼저 시도하였습니다.
- 해당 모델은 Question의 정답이 들어있는 Passage는 Positive, 이 외의 Passage는 Negative로 설정하여 Positive Passage와 Question은 가까워지게, Negative Passage와 Question은 멀어지게 학습합니다.



- 이론적 근거

Vladimir Karpukhin et al.에서는 문서를 dense embeddings로 mapping하는 방법을 제안하며, sparse embeddings으로 mapping하는 TF-IDF나 BM25 방식보다 top-20 passage retrieval 정확도가 9%p ~ 19%p의 향상을 보인다는 사실을 보고 하고 있어 해당 논문의 방법론을 통해 본 프로젝트의 retriever 성능을 높이고자 했습니다.

- 학습 설정

- 인코더의 체크포인트로는 `klue/bert-base`, `klue/roberta-base`를 사용하였으며, `epochs` = 20 ~ 25, `lr` = 1e-5, `in-batch negative samples` = 7 등의 값을 사용하여 파인튜닝을 진행하였습니다.

- 구현 결과

- Query가 변화하여도 top-k passages 구성이 크게 변화하지 않는 retriever를 구현하게 되었습니다. 예컨대 “유령은 어느 행성에서 지구로 왔는가?”라는 질의와 유사하다고 판단하는 지문으로는 유니코드 관련 지문, 동계 올림픽 기간 동안 통계 집계를 표 형식으로 정리한 지문, 비트겐슈타인 관련 지문 등의 순서로 검색(retrieval) 결과를 내놓았습니다. 이때 각 지문은 백슬래시를 포함한 특수문자를 상당히 많이 포함했었습니다.
- Retrieval 시에는 Faiss 라이브러리를 사용하여 scaling까지 고려한 구현을 하였고 실험을 진행해보았는데, Faiss의 indexer를 활용한 검색에서 뽑힌 top k개의 지문 간에는 일정한 공통점이 있는 것을 발견하였습니 다. 예컨대 k 값을 바꾸거나 dense embeddings를 인코더의 하이퍼파라미터를 바꿔서 뽑아보니 특수문자가 다수 포함되거나, 특정 단어(- 초등학교)가 반복해서 등장하는 지문만 골라내주는 등의 결과를 보여주었습 니다.

- 오류 원인 분석

- 오류를 디버깅 하기 위해 구현 코드에서 line by line 논리적인 오류를 찾아내려는 방식과 학습이 잘 되었던 참고 코드에서 데이터셋, 실험 조건, 리팩토링 등 변경사항을 추적하는 방식, 크게 두 가지 방향으로 접근하였 습니다. 전자의 방법을 통해서는 정해진 기간 내에 치명적인 논리적 오류를 찾아내지 못 하였습니다. 후자의 방법을 통해서는 큰 규모의 데이터셋을 학습시킨 bi-encoder는 sparse retriever보다 top-k accuracy가 우 수한 것을 몇 가지 샘플에서 확인하였지만, 보다 작은 규모의 데이터셋을 학습시켰을 때는 dense retrieval이 의도된 방향과 크게 벗어나는 쪽으로 작동하는 것을 확인했습니다.

Reader

Data Augmentation

- 외부데이터 활용이 가능하며, KorQuAD dataset를 대회에 활용하였습니다.

- KorQuAD datasets
 - train dataset : 60,407건
 - validation dataset : 5,774건
- KorQuAD dataset를 활용한 방법은 다음과 같습니다.

	train num_rows	validation num_rows	EM Score	F1 Score
baseline dataset (증강 X)	3952	240	67.0833	74.8638
train dataset만 증강	64205	240	69.5833	78.0053
train dataset 중, 중복된 context를 제외하고 증강	13558	240	69.1667	78.5215
train, validation dataset 모두 증강	64205	6004	87.7248	92.4542

- 외부 데이터를 활용하였을 때, reader의 성능이 개선되는 것을 확인하였습니다.
- train, validation을 모두 증강하였을때 reader의 성능이 가장 높았으나, retrieval module과 연결 한 후, inference 성능이 개선되지 않았습니다.

Curriculum Learning

- 모델이 잘 예측하지 못하는 긴 답변에 대한 예측 결과를 개선하기 위하여, **모델 학습 시 쉬운 데이터부터 어려운 데이터 순서로 데이터를 사용**했습니다.
- validation set에 대한 inference 결과를 확인해본 결과, 오답 데이터들은 전체 데이터셋에 비해 answer와 context의 길이가 길다는 특징을 가지고 있습니다.

	answer 평균값	answer 중앙값	context 평균값	context 중앙값
전체 데이터	4.066667	3.000000	16287.641667	15743.500000
오답 데이터	4.602941	3.000000	16514.176471	15868.000000

- 특히 answer의 경우, 중앙값은 같지만 오답 데이터에서의 평균값이 0.5 정도 높은 것으로 보아, answer의 길이가 매우 긴 이상치 데이터에 대해 모델이 정확하게 예측하지 못하고 있음을 확인할 수 있습니다.
- 이에 따라 모델 학습 시 데이터를 쉬운 것부터 어려운 것 순서로 제공하기 위하여 answer와 context의 길이로 정렬한 데이터로 학습을 진행하였습니다.

- 실험 결과

	use sorted data	train_val_EM	train_val_F1	inference_val_EM	inference_val_F1
baseline	x	71.25	78.397	60.417	66.533
Curriculum Learning	o	67.083	76.43	53.75	61.116

- reader 모델의 evaluation 결과 EM과 F1 모두 하락하였으며, inference 결과 또한 크게 하락하였습니다.
- 모델을 학습할 때에는 **일반적으로 데이터셋을 무작위로 섞어서 사용**함으로써 일반화 성능을 향상시키고, 데이터의 반복적 패턴을 학습시키는 것을 예방하는 동시에 편향을 최소화하는 효과를 가질 수 있는데, 이러한 무작위성을 제거하고 **의도적으로 경향성을 가지는 데이터를 통해 모델을 학습시켰기 때문에 성능이 하락한** 것으로 볼 수 있습니다.

Training with Retrieved Context

- reader model을 train할 때의 validation set에 대한 evaluation 결과와, validation set에 대한 inference 결과에 유의미한 차이가 있어, 이 둘 사이의 극간을 줄이 기 위하여 **reader model의 학습 시에 context로 golden passage와 더불어 retriever가 추출한 상위 3개의 passage를 함께 전달**하였습니다.
- reader model의 training 과정에서는, validation set에 대하여 golden passage로부터 정답을 찾아내는 식으로 evaluation을 진행합니다.
 - 이 때의 validations set에 대한 EM과 F1은 각각 `71.25`, `78.397` 이었습니다.
- 전체 inference 과정에서는, retriever가 추출해온 top k개의 passage를 이어붙인 context로부터 정답을 찾아내는 식으로 test set에 대한 prediction을 진행하고, 마찬가지로 방식으로 validation set에 대하여 evaluation을 진행합니다.
 - 이 때의 validations set에 대한 EM과 F1은 각각 `60.417`, `66.533` 이었습니다.
- BM25 retriever가 95% 이상의 정확도를 가짐에도 이와 같은 차이가 존재하는 이유는, training 과정에서는 reader 모델이 평균 500 token의 짧은 golden passage에서 정답을 찾는 식으로 학습을 진행했지만, inference 과정에서는 평균 16000 token의 길고 noise가 포함된 passage에서 정답을 찾기 때문에 이 두 방식 사이의 극간으로 인해 성능이 저하된 것으로 생각해볼 수 있습니다.
- 따라서 reader model의 학습 시에 golden passage만을 사용하는 대신, golden passage와 retriever가 추출한 golden passage가 아닌 context를 함께 사용하면, **training 과정과 inference 과정에서의 context를 비슷한 형태로 맞추어줌**으로써 성능 향상을 도모해볼 수 있을 것이라는 가설을 바탕으로 실험을 진행했습니다.

- 실험 결과

	context	LB_EM	LB_F1	train_val_EM	train_val_F1	inference_val_EM	inference_val_F1
baseline	golden passage	63.7500	74.1600	71.25	78.397	60.417	66.533
Training with Retrieved Context	golden passage + retrieved context	63.7500	76.5600	69.167	77.809	65.0	72.863

- baseline에 비해 길고 noise를 포함하는 context로 학습하였기 때문에 학습 시 evaluation 결과는 EM과 F1 (train_val_EM, train_val_F1) 모두 소폭 하락하였습니다.
- 하지만 validation set에 대해 inference를 진행한 결과, EM은 60.417 에서 65.0 으로, F1은 66.533 에서 72.863 으로 크게 개선된 것을 확인할 수 있습니다.
- 더불어 **evaluation에서의 결과와 inference에서의 결과 사이의 격차가 크게 감소**하여(EM: 10.833 → 4.157 , F1: 11.864 → 4.946), 학습 시와 추론 시의 input의 형태를 비슷하게 만들어주는 것이 모델의 성능 개선에 유의미한 변화를 야기함을 확인할 수 있습니다.

Ensemble

Hard Voting

- inference 결과물로 만들어진 predictions.json 파일을 활용하여, 예측된 answer 중 가장 많이 등장한 답을 선택하는 방식으로 앙상블을 진행하였습니다.
- 최빈값이 여러 개인 경우, 최빈값 중에서 무작위 선택하였습니다.


Soft Voting

- inference 결과물로 만들어진 nbest_predictions.json 파일을 활용하여, 예측된 후보 answer 확률의 평균으로 앙상블을 진행하였습니다.


References

Elasticsearch와 Kibana의 개발사인 Elastic에 오신 것을 환영합니다

Elastic은 데이터 검색, Observability 및 보안 솔루션을 제공하여 팀이 인프라와 앱을 완벽하게 파악할 수 있도록 지원합니다.

 <https://www.elastic.co/kr/>

Search. Observe. Protect.

 elastic

3. 프로젝트 고찰

잘한 점 및 아쉬운 점

잘한 점

- 도메인과 베이스라인 코드 이해에 어려움을 겪었지만 적응하고 소신있게 다양한 시도들을 했습니다.
- 이전 프로젝트에서 아쉬웠던 git 협업 방식을 보완하기 위해 git flow 방식을 채택하여 적용하였습니다.


아쉬운 점

- 실험에 사용되는 configuration 관리, wandb 및 validation inference 등 실험을 위한 pipeline의 설계가 이전 프로젝트에 비해 부족하여 실험 결과를 분석하는데 어려움이 있었습니다.
- Inference 결과를 사후분석하고 이를 바탕으로 새로운 실험을 설계하는 논리적인 과정이 부족했습니다.
- 최종 answer를 추출할 때, retriever가 추출한 context의 중요도를 고려하지 못했습니다.
- 구현에만 집중하고, 구현된 코드로 치밀하게 실험을 진행하지 못했습니다.
- 프로젝트 설계가 치밀하지 못 한 것이 성능 개선을 많이 하지 못 한 가장 큰 원인이라는 것을 느낀 프로젝트였습니다. 예컨대 어떠한 시도를 하는 데 앞서서 여러 가지 시도들에 대해 예상되는 비용과 그 결과로서 얻을 수 있는 성능 개선 결과를 비교하고, 가장 효율적인 순서대로 구현을 하는 방법으로 선택을 해야 했지만 그러지 못 한 것이 아쉬웠습니다.

개인 회고

▼ 문지혜

학습목표 & 수행한 것

-  ODQA를 이해하고, Reader와 Retriever의 성능을 개선하자!

- 외부 데이터를 도입 - KorQuAD
 - baseline으로 주어진 데이터 이외에 다른 데이터셋을 이용해 보고자 KorQuAD dataset을 도입했다. KorQuAD dataset을 도입하기 위해 세가지 방법을 시도해 보았다.
 - huggingface dataset 이용 : KorQuAD dataset과 base data를 concat하여 huggingface에 커밋했고, 이를 활용하려고 하였으나. baseline code와 huggingface dataset을 호환하지 못해 실패하였음.
 - pyarrow file 만들기 : KorQuAD dataset과 base data를 concat하여 pyarrow file로 만들고, 이를 모델에게 feed 하는 방법을 시도하였으나, pyarrow의 메타데이터가 제대로 만들어지지 않아 실패
 - base data는 그대로 두고, dataset 라이브러리의 load_dataset으로 KorQuAD 불러와 코드 내부에서 합쳐주었음.
- 앙상블 코드 구현
 - 대회 마지막 날, 실험 결과를 앙상블 하여 점수를 높이고자 노력하였다. 나는 soft voting 방법을 통해 앙상블을 구현하였다.
 - inference를 실행하면, nbest_prediction.json 파일에 질문마다 answer가 될 수 있는 후보 top 20을 결과물로 만들어준다. 이 파일을 활용하여 각 질문에서 나올 수 있는 답 후보들을 key로 설정하여 딕셔너리를 만들고, value에는 리스트 형태로 답 후보들의 확률을 append 하였다.
 - 이렇게 만들어진 answer 후보에 대한 확률 리스트를 딕셔너리 형태로 가지게 되면, 리스트의 평균을 산출하여 각 answer에 대한 평균 확률로 최종 답안을 선정하게 된다.

마주한 한계 & 아쉬웠던 점

- 긴 코드를 이해하는데 시간이 오래 걸린다.

긴 코드를 이해하고 디버깅을 하는 과정이 있었는데, 내가 코드 이해력이 낮아서 시간이 오래 걸렸다. DPR에 대해 잘 이해했다고 생각했는데, DPR 코드를 파악하고 문제를 발견하는 것이 어려웠다. 앞으로 긴 코드를 보더라도 빠르게 상황을 파악하고 디버깅을 할 수 있도록 해보자!
- 선택과 집중을 하자

이번 대회는 최종 프로젝트과 동시에 고민해야 했기 때문에 집중이 흐트러진 것 같다. 나는 무언가를 결정할 때 오래 고민하는 스타일이기 때문에 두 가지 일을 동시에 하려면 서로 다른 일로 전환하는데 적응시간이 필요하다. 그런데 이번에는 어쩔수 없이 두 프로젝트 모두 생각해야 했었어서 집중이 모자랐다. 앞으로 이러한 상황이 다시 발생하면, 그냥 과감히 우선순위를 정하거나 시간을 나눠 일을 해야겠다.

▼ 박지은

1. 학습 목표

- git branching 전략으로 기존의 git flow가 아닌 github flow를 활용해보기
 - 프로젝트의 특성 상 git flow가 더 적절한 전략이기는 하지만, 최종 프로젝트때 보다 능숙하게 github flow를 활용할 수 있게 연습삼아 github flow를 사용해보기로 하였다.

- main branch 이전에 예비 branch로 develop branch가 하나 추가된 것 외에 큰 차이는 없었지만, main branch 위에 develop branch를 올리고, 그 위에 feature branch들을 올려가는 과정을 연습해보았다는 점에서 의미가 있었다는 생각이 든다.
- 기존보다 체계적으로 실험 파이프라인 설계하기
 - wandb에 평가 metric의 log를 연결하고, validation set에 대한 inference 결과와 그를 이용한 사후 분석 코드까지 자동으로 실행되도록 하는 것이 목표였으나, baseline code에 대한 이해가 늦어져 validation set에 대한 inference 결과를 추출하는 것까지만 실천할 수 있었다.
 - 지난 대회들과 다르게 config.json, config.yaml로 실험과 관련한 인자들을 조절하는 것이 아닌, huggingface의 TrainingArguments와 유사하게 만든 class들을 통해 인자를 조절하는 방식으로 baseline code가 설계되어 있었는데, 이에 적응하는 것에 시간이 걸려 ‘실험명’과 관련한 자동 설정 없이 대회를 진행했다. 이로 인해 실험명을 수동으로 일일이 정리해두었어야 하는데, 이 점이 생각보다 시간과 에너지 소모가 컸던 것 같다. 이전에는 매번 시간이 조금 걸리더라도 실험을 위한 파이프라인을 꼼꼼하게 설계하고 실험을 시작했었기 때문에 오히려 그 중요성을 체감하지 못했었는데, 이렇게 당연했던 것을 당연하게 하지 않았던 경험을 통해 실험 파이프라인을 꼼꼼하게 설계하는 것의 중요성을 깨달을 수 있었다.
- 데이터와 모델 중 어느 쪽으로 치우치지 않는 다양한 실험 진행해보기
 - 이전까지의 대회에서 데이터와 관련한 부분을 많이 다뤄왔던 것 같아서 이번 대회를 진행하면서는 모델과 관련한 실험도 적극적으로 도전해보아야겠다는 생각을 했었다. 데이터에 비해서는 아직 논리적인 사고과정을 통해 모델을 선택하고, 코드를 작성하여 이를 활용하는 것이 부족하다고 생각했기 때문이다.
 - 이번 대회를 진행하면서도 아예 새로운 모델을 구현해보지는 못했지만, BM25를 적용하고 Trainer의 method를 override하여 curriculum learning을 실험하는 등 이전 대회에 비해서는 모델 쪽과 관련한 실험을 많이 해본 것 같다.
 - 모델과 엔지니어링 쪽에서 더 많은 기회가 주어질 파이널 프로젝트를 통해 이번 대회에서의 아쉬움을 메꾸어갈 예정이다.

2. 프로젝트 진행 내용

- EDA
 - retrieval에 사용되는 wikipedia 말뭉치와, reader의 학습에 사용되는 데이터셋에 대한 EDA를 진행했다.
 - 전에 다루어보았던 범주형, 수치형 데이터를 거의 포함하고 있지 않은 데이터이다보니 어떤 식으로 EDA를 진행하는 것이 좋을지 고민하는 시간이 있었고, 대체로 text 길이의 분포에 대한 분석을 진행했다.
 - 전처리를 위해 데이터가 특수문자 및 외국어를 포함하는지 여부 및 그 비율에 대한 분석도 진행했었는데, 상당수의 정답에도 특수문자와 외국어가 포함되어있어 이를 단순히 제거해버리는 것은 올바른 방향이 아니라고 생각했다. 그래서 특별한 preprocessing 없이 모델링을 진행했었는데, 대회 막바지에 preprocessing을 한 데이터로 학습한 모델의 성능이 확 오르는 모습을 보면서 EDA 당시에 조금 더 꼼꼼하게 고민했어야 했다는 생각이 들었다.
- BM25 retriever
 - 강의 내용에 포함되어있었던 BM25를 더 알아보고 TF-IDF의 수식과 비교해보면서, TF-IDF에 비해 BM25가 가지는 이점에 대해 생각해보는 시간을 가졌다. 단순히 좋은 알고리즘이라니까 가져다 써보기에 그치지 않고 알고리즘에 대한 이해를 바탕으로 task에 적용해보았다는 점에서 성장한 스스로의 모습을 체감할 수 있었다.
- Data Augmentation
 - 데이터도 기존 대회들에서는 주로 csv 형식의 파일을 huggingface datasets repo에 올려 사용했었는데, 본 대회에서는 json이나 .arrow 형식의 파일을 주로 다루게 되어 초반에 적응하는 시간이 조금 필요했었다.
 - 기존 데이터에 KorQuAD 데이터를 추가하는 식으로 augmentation을 진행할 때에도 파일 형식으로 인한 어려움을 겪었었는데, Datasets과 DatasetsDict 객체에 대한 이해를 바탕으로 문제를 해결해본 경험이 재밌었다.
- Curriculum Learning
 - 내 생각에는 논리적인 가설로 설계한 실험이었지만 생각한대로 성능이 개선되지 않아 그 이유를 찾아보는 일련의 경험을 해볼 수 있어서 유의미한 실험이었다. 경향성을 가지는 데이터로 학습한 모델이 가지는 편향성과 취약성에 대해 경험적으로 배워볼 수 있었다.
 - 또한 curriculum learning을 구현하기 위하여 공식 github, docs를 통해 Trainer class를 분석하고, 관련 method를 override해보는 경험을 해볼 수 있어서 좋았다.
- Training with Retrieved Context
 - training 과정과 inference 과정을 유사하게 맞춰주기 위하여 고안한 방법이었고, 성능 향상까지 이뤄볼 수 있었다.
 - 다만 reader 모델을 수정하여 training 과정과 inference 과정의 극간을 채워주는 것도 좋지만, retriever를 수정하는 접근 또한 시도해보았으면 좋았을 것이다.
 - 특히 retriever가 뽑아온 30개의 context를 단순히 concat해서 사용하는 것보다는 context를 각각 reader 모델에 전달하는 방법과, retriever가 추출한 context의 순위를 반영하는 방법을 시간 부족으로 실험으로까지 연결하지 못했던 점이 아쉽다.

3. 협업과 커뮤니케이션

- Github
 - 4번의 대회를 거치며 도입한 Github을 통한 협업 방식(issue, PR, branching 전략 등)에는 잘 적응했다.
 - 베이스라인 코드의 항상성 유지 여부를 체크하는 작업의 필요성을 깨달았고, 이에 따라 이후의 대회 및 프로젝트에서 코드의 유지/보수를 위하여 Github Action을 도입해보아야겠다는 생각을 했다.
 - issue와 PR을 통한 커뮤니케이션의 비율을 높여보면 좋을 것 같다는 피드백을 받아볼 수 있었다. 회의에서 직접적으로 공유할 내용을 최소화할 수 있도록 github을 적극적으로 활용하고 확인하는 습관을 들여야겠다.
- PM으로서의 나
 - 학교에서도 부스트캠프에서도 느끼지만 나는 어떤 팀의 구성원으로서 가져야 할 커뮤니케이션 및 문제 해결 역량 면에서는 꾸준히 성장하고 있는 것 같은데, 리더로서 가지면 좋을 마음가짐에 있어서는 한참 부족하다는 생각이 든다. 아니면 어쩌면 리더가 마땅히 가져야 할 책임감이 아직 나에게는 너무 무겁게 느껴지는 것일 수도 있겠다. 이것도 노력하다보면 언젠가 편해지는 날이 올 것이라고 믿고 싶다!

▼ 박경택

0. 프로젝트 개요

- 주어진 질의와 지문을 이해하고 답변을 추론하는 기계 독해 인공지능 만들기
- 특히 이번 프로젝트는 단순 기계 독해 태스크라기보다는 오픈 도메인 질의응답이라고 표현하는 것이 더 적절한 태스크.

1. 목표 설정 및 수행 사항

논문 구현 수행

논문 내용을 이해한 것을 바탕으로 바닥부터 코드로 구현해내는 경험을 해보고자 하였다. 인공지능 산업은 매우 빠르게 발전하는 만큼 새로운 내용을 빠르게 습득하고 실험해보는 게 중요할 것인데, 새 논문을 빠르게 읽어내는 능력과 이를 정확하고 빠르게 구현하는 능력 두 가지가 중요할 것 같다는 생각에서였다.

다만 본 프로젝트에서 구현해보고자 한 것은 DPR (dense passage retrieval) 기술로, 약 3년 전인 2020년에 제안된 방법론이다. 사실 최신 내용이라고 볼 수 없지만 본 기술을 구현한 것을 바탕으로 retrieval 구현 방식의 변형을 가한 논문을 읽고 구현해보려는 계획이 있었다. 지문과 질의를 각각 bi-encoder로 맵핑하는 방식 외에도 ColBERT와 같은 기술을 쓸 수도 있었다. 이번에는 기존 코드 등을 조합하고 활용해보는 것 이상으로 처음부터 구조를 설계하고 직접 구현하고자 하는 목표도 있었다.

2. 결과와 분석

구현 결과 및 분석

Bi-encoder 기반으로 학습하여 dense embeddings로 맵핑한 뒤 Faiss 라이브러리로 임베딩을 빠르게 찾는 일련의 파이프라인을 구현하였다. 실행 수준에서의 오류는 없었으나, 논리적인 오류는 있었던 것으로 파악된다. 그 구체적인 내용을 파악해보자면, 질의가 변화하여도 이와 관련된 상위 k개의 지문 구성이 크게 변화하지 않는 검색기(retriever)가 구현되었다.

예컨대 “유령은 어느 행성에서 지구로 왔는가?”라는 질의와 유사하다고 판단하는 지문으로는 유니코드 관련 지문, 동계 올림픽 기간 동안 통계 집계를 표 형식으로 정리한 지문, 비트겐슈타인 관련 지문 등의 순서로 검색 결과를 내놓았다. 이때 각 지문은 백슬래시를 포함한 특수문자를 상당히 많이 포함했었다. 또 다른 실험에서는 질의(query)를 어떻게 주든 상위 k개의 지문 구성에 모두 “-초등학교”로 끝나는 지문이 다수 포함되는 결과를 얻게 되었다.

Faiss를 활용하지 않고 유사도를 평가하는 방법을 사용해도 위와 비슷한 결과를 내놓았던 것으로 보아 bi-encoder 부분에서 문제가 있었다고 판단했고, 데이터 문제인지 모델 구현 레벨에서의 문제인지 등을 파악해보고자 했으나 주어진 시간 내에 디버깅을 완수하지는 못 하였다.

3. 🤔 한계와 🗣️ 교훈, 📩 피드백

이번 프로젝트를 회고하면서 자원 활용 능력의 부족이 가장 큰 문제점이었다고 생각된다. 좋은 자원, 환경이 있었음에도 이를 100% 활용하지 못 하였다. 예컨대 디버깅을 할 때 VS code의 디버거를 능숙하게 활용했다면 디버깅 시간을 훨씬 줄일 수 있었지만 제대로 사용하지 못 하였다. 사실 해당 기능은 시도를 여러 차례하였고 관련 내용을 숙지하는 데 시간을 할애하였는데, 매 실험마다 몇 시간 단위로 이루어지는데 눈에 보이는 에러를 처리하는 데 급급하다가 시간을 허비한 게 컸던 것 같다. 시간이 지체될 때는 멘토링을 적극적으로 활용할 생각을 하지 못 했던 것도 큰 파이를 차지했다는 생각이 들었다. 어떻게 잘 질문할지 등에 대한 고민이 부족했던 것 같다. 퍼포먼스 부족에는 다른 이유도 있었겠지만, 위 사항이 가장 근본적인 원인이었다고 생각한다.

또한 의존성 관리, 디렉토리 설정과 같은 변수를 어디에 정의하고 어떻게 불러오는지 등에 있어서도 시간 소모를 많이 하였다. 프로젝트 root 폴더 혹은 root/code 폴더 이외의 경로 어디에서도 원하는 코드 실행 결과를 얻고자 했지만 이를 반영하기 위해서는 코드 디렉토리 전반적으로 상대 경로를 수정해야 하는 등의 번거로움이 필요했던 것 같다. 프로젝트의 목표 달성을 위해 부차적인 요소일 수 있는 데 집착하게 되는 것 또한 시간을 효율적으로 관리하지 못 하는 것의 원인이 되지 않았나 싶다.

▼ 윤지환

🔨 목표와 시도 🔨

최소한의 목표들

- 우선 코드가 복잡해서 가능한 모두 일일이 뜯어보고 이해해보고자 했다.
- 현재 많이 쓰이는 좋은 모델들을 찾아서 적용시켜보고자 했다.
- Reader 모델의 결과물들을 직접 확인해보며 Retrieval 모델과의 상호작용에서 어떤 부분을 수정해야 할지 점진적으로 개선해보는 방향을 잡고자했다.

복잡한 베이스라인

- Reader와 Retrieval 두 단계로 모델링이 이루어진 상당히 많은 양의 베이스라인 코드를 이해하는데 많은 시간이 들었다.
- 베이스라인 코드를 실험하기 편하게 **리팩토링해보며 전체적인 구조를 파악**하는데 도움이 많이 되었다.
- 특히 딥러닝 프로젝트 깃 저장소에서 많이 보이는 **셸 스크립트 형태로 통제하는 실험법**이 어떻게 돌아가는 것인지 이번 기회에 알게된 점이 좋았다.

Elasticsearch Retrieval

- Elasticsearch**라고 하는 분산검색 시스템을 가져와 기존 모듈에 추가했다.
- Retrieval 자체는 잘되었으나 기본 bm25 알고리즘과 속도면에서 거의 차이가 없었던 것이 아쉬웠다. 같은 조건에서 오히려 bm25보다 성능이 약간 떨어지기도 했다.
 - 구동 코드상에서의 문제라기보단 elasticsearch가 실행되는 순서와 환경적인 문제였다.
- 단일 MRC 모델에서 **가장 높은 성능**을 보인 방법이기도 했다. (**EM 63.75 → EM 67.5**)
 - 기존과 다른 점은 개행문자, #,과 같은 **의미없이 반복되는 문자를 제거**해준 것이었다.
 - wiki document와 reader train 데이터 **모두 처리**해주어야만 성능이 오른다는 점을 보고 ODQA에선 **reader와 retrieval 모델 사이에 맞는 조합이 존재**함을 깨달았다.

😎 배움과 아쉬움 😎

다른 조들의 다양한 실험들

- 마스터 클래스에서 다른 조들이 굉장히 다양한 시도들을 한 것을 보고 감명 받았다.
- ColBERT, custom 지표, MRC tag와 같이 실험 방법에 대한 변화 뿐만 아니라 다양한 모델, input 형태를 차용한 **꺾이지 않는 실험정신**이 중요한 것 같았다.
- 많이 실험하고 많이 실패한 노력의 흔적들이 보여서 배워야겠다고 생각했다.

다양하고 논리적인 실험의 부재

- 네 번의 대회 중 마지막으로 **지금까지 쌓은 경험과 실력**을 녹여내야 했을 대회에서 **가장 불만족스러운 운영**을 했던 것 같다.
- 의미있는 시도들이 아닌 하이퍼파라미터 튜닝이나 주어진 기능의 수치를 바꾸거나 사롭지만 유명한 방법(ex. elastic retrieval)을 추가해보고 확인하는 수준에서 그쳤다.
- 무엇보다 기본적인 **실험 이후의 사후 분석**을 통해 **다음 실험을 위한 논리적인 전개**가 부족했다.

통제되지 않는 실험 세팅

- 기존과 달리 이번엔 두 개의 모델, 문서를 가져오는 retrieval과 가져온 문서에서 답을 추출해내는 reader **두가지를 동시에 고려**해야하는 문제였다.
- 때문에 한 가지 변인만 변화시키고 나머진 그대로 통제하는 방식이 먹히지 않았다.
 - 따라서 실험이 있을 때마다 최적의 성능을 위한 하이퍼파라미터 조합을 찾아야했다.
- 이런 문제를 실제 업무에서 겪게된다면 효율적으로 해결할 수 있는 방법이 무엇이 있을지 고민해볼 계기가 되었던 것 같다.

🚢 앞으로의 지향점 🚢

- 나 스스로가 일정에 있어서 후반부로 갈수록 **자꾸 늘어지고 의욕을 서서히 잃어가는 고질적인 문제점**이 있음을 알았다. **반복적인 동기부여**와 흥미를 잃지 않기 위해 **나만의 분석 스토리**를 만들어가야겠다.
- 집중력**이 예전에 비해 많이 떨어진 것을 느낀다. 집중하는 환경을 다시 한번 점검해야겠다.
- 많은 **논리적인 이유가 있는 실험**들과 그것들을 **체계적으로 관리**하는 방법을 정립해야겠다. 이번 대회에선 많이 부족했는데 **부족한 만큼 최종 프로젝트에서 채워보고자 한다.**
- 프로젝트의 한 단위기간 동안 일부로라도 **쉬는 팀**을 만들어야겠다. 100% 모든 시간을 투입하는 것이 **무조건 좋은게 아니**라는 것을 깨달았다.

▼ 송인서

1. 목표 및 자가평가

- 주도적으로 참여하기
 - 이전 대회들에서 매번 다음 대회부터는 주도적으로 참여하겠다고 다짐하였으나 이번에도 큰 개선을 하지 못한 것 같아 아쉬움
 - 이번엔 다른 대회들보다도 전반적인 아웃라인을 명확하게 그려내지 못한 것 같음
- 직접 수행할 사항 결정하고 결과 만들기
 - 직접 수행할 사항을 결정하지 못한 것이 대회에 대해 제대로 파악하지 못했다는 증거라고 생각됨
 - 하나라도 자발적으로 수행하고 팀원에게 기여하는 경험을 했어야 스스로에게도 작은 전환점이 될 것 같은데 스스로 태스크를 너무 막막하게 여기는 것이 아닌가 하는 생각이 들
 - 해야지 하고 마음먹고 나서도 혹시 완수하지 못할까봐 팀원들에게 제대로 선언하지 못하니 호지부지 된 것 같음

2. 프로젝트 수행 내용

- Hard Voting 구현
 - 인퍼런스 결과를 사용해 Hard voting을 수행하는 코드 작성
 - 최대한 간결하게 작성하고자 노력했음

- 최빈값이 여러개인 경우 최빈값들 중에서 무작위로 선택하는 방법을 택함
- 인퍼런스 결과에 대해 사람이 확인할 때에는 큰 차이가 없으나 특수문자 등이 다르게 포함되어 다르게 카운팅하는 문제에 대해 고민해보았음

3. 한계와 교훈

- 스스로 하겠다고 마음먹고 나면 다른 팀원들에게 '제가 하겠습니다'라고 선언하고 스스로에게 압력을 가해야 하겠음



공개선언효과 (Public Commitment Effect, 떠벌이효과) 라는 심리학 효과가 있음. 공개적으로 계획과 목표를 공표하면 결심을 끝까지 지킬 확률이 높아지는 효과. 인간의 본능상 떠벌리면 결과로 보여주기 쉬워진다는 것임. 한국처럼 체면이 중요한 사회에선 그 효능이 훨씬 높다는 연구결과도 있다!



선언형 프로그래밍: "일단 선언하고 나면 개발하게 돼 있어요", 그러니까 동네방네 떠돌고 다니라는 뜻

- 동료들 좀 더 믿기
 - 지금 다른 팀원들을 불신한다는 것이 아니라, 혼자 하는 것보다 동료와 함께 하는 것이 문제해결력이 훨씬 높다는 사실을 언제나 늘 생각하기
 - 늘 감사하고, 지켜봐주셔서 감사합니다. 한사람분의 인간이 되겠습니다.

 박경택

 박지은

 문지혜

 윤지환

 송인서