Dialogue Summarization I 일상 대화 요약

학교 생활, 직장, 치료, 쇼핑, 여가, 여행 등 광범위한 일상 생활 중 하는 대화들에 대해 요약합니다.

#비공개대회 #AI부트캠프14기 #NLPAdvanced

베이스라인 코드

데이터 다운로드 링크

https://aistages-api-public-prod.s3.amazonaws.com/app/Competitions/000365/··· 복사 | 다운로드

코드 구조

코드 파일은 아래와 같이 구성되어 있습니다.

```
code (폴더)
├── baseline.ipynb
├── config.yaml
├── requirements.txt
└── solar_api.ipynb
```

code 폴더에는 대회에서 사용될 baseline code인 baseline.ipynb, 베이스라인 코드 실행에 필요한 기본적인 패키지 정보들을 모아둔 requirement.txt, 그리고 베이스라인 모델의 매개변수 정보들을 모아둔 config.yaml이 들어있습니다.

코드 설명 (baseline.ipynb)

1. 데이터 가공 및 데이터셋 클래스 구축하기

- 데이터셋을 불러와 BART 모델의 encoder와 decoder의 입력 형태로 가공해줍니다.

- 가공된 데이터를 torch dataset class 로 구축하여 모델에 입력가능한 형태로 만듭니다.

1.1. `class Preprocess`

: 데이터 전처리를 위한 클래스로, 데이터셋을 데이터프레임으로 변환하고 인코더와 디코더의 입력을 생성합니다.

1.2. `class DatasetForTrain, DatasetForVal, DatasetForInference`

: Train, validation, test에 사용되는 Dataset 클래스를 정의합니다.

1.3. Define `prepare_train_dataset`

: Tokenization 과정까지 진행된 최종적으로 모델에 입력될 데이터를 출력합니다.

: encoder의 input 데이터와 decoder의 input, ouput 데이터를 생성합니다.

```
tokenized_encoder_inputs = tokenizer(encoder_input_train, return_tensors="pt", padding=True,
                          add_special_tokens=True, truncation=True, max_length=config['tokenizer']['encoder_max_len'], return_token_type_ids=False)
tokenized_decoder_inputs = tokenizer(decoder_input_train, return_tensors="pt", padding=True,
                         add_special_tokens=True, truncation=True, max_length=config['tokenizer']['decoder_max_len'], return_token_type_ids=False)
tokenized_decoder_ouputs = tokenizer(decoder_output_train, return_tensors="pt", padding=True,
                         add_special_tokens=True, truncation=True, max_length=config['tokenizer']['decoder_max_len'], return_token_type_ids=False)
```

2. `Trainer` 및 `TrainingArguments` 구축하기

- Huggingface 의 `Trainer` 와 `TrainingArguments` 를 활용하여 모델 학습을 일괄적으로 처리해주는 클래스를 정의합니다.

2.1. Define `compute_metrics`

: 모델 성능에 대한 평가 지표를 정의합니다. 본 대회에서는 ROUGE 점수를 통해 모델의 성능을 평가합니다.

2.2. Define `load_trainer_for_train`

: 학습을 위한 Trainer 클래스와 매개변수를 정의합니다.

2.3. Define `load_tokenizer_and_model_for_train`

: 학습을 위한 BART 기반의 tokenizer와 사전 학습된 모델을 불러옵니다. 본 베이스라인은 한국어에 특화된 KoBART (https://huggingface.co/digit82/kobart-summarization)를 사용합니다.

: tokenizer를 정의할 때, 도메인마다 문장에서 자주 등장하는 고유 단어들이 분해되는 것을 방지하기 위하여 special_tokens을 지정해줍니다.

3. 모델 학습하기

- Trainer 클래스를 불러온 후 모델 학습을 진행합니다.

```
# 사용할 모델과 tokenizer를 불러옵니다.
generate_model , tokenizer = load_tokenizer_and_model_for_train(config,device)
print('-'*10,"tokenizer special tokens : ",tokenizer.special_tokens_map,'-'*10)

# 학습에 사용할 데이터셋을 불러옵니다.
preprocessor = Preprocess(config['tokenizer']['bos_token'], config['tokenizer']['eos_token'])
data_path = config['general']['data_path']
train_inputs_dataset, val_inputs_dataset = prepare_train_dataset(config,preprocessor, data_path, tokenizer)

# Trainer 클래스를 불러옵니다.
trainer = load_trainer_for_train(config, generate_model,tokenizer,train_inputs_dataset,val_inputs_dataset)
trainer.train()  # 모델 학습을 시작합니다.
```

[2750/5000 20:11 < 16:32, 2.27 it/s, Epoch 11/20]

Epoch	Training Loss	Validation Loss	Rouge-1	Rouge-2	Rouge-l
1	5.712300	2.528442	0.184872	0.035078	0.179494
2	1.265600	0.636261	0.358953	0.120937	0.341066
3	0.591500	0.585961	0.370596	0.130791	0.354864
4	0.535200	0.574670	0.374552	0.134506	0.357354
5	0.503300	0.568946	0.375335	0.137726	0.360847
6	0.479600	0.565534	0.378215	0.138479	0.362137
7	0.459200	0.564776	0.374650	0.137188	0.359153
8	0.440900	0.563768	0.386005	0.144035	0.370481
9	0.425200	0.566193	0.378511	0.141013	0.362608
10	0.411600	0.568887	0.376012	0.140056	0.361476
11	0.400000	0.567971	0.384345	0.144940	0.370093

4. 모델 추론하기

4.1. Define `prepare_test_dataset`

: Tokenization 과정까지 진행된 최종적으로 모델에 입력될 데이터를 출력합니다.

: 학습 과정에서와는 다르게 추론 시에는 decoder의 output 데이터를 생성할 필요없이 encoder와 decoder에 입력될 데이터만 생성합니다.

```
test_tokenized_encoder_inputs = tokenizer(encoder_input_test, return_tensors="pt", padding=True,
                add_special_tokens=True, truncation=True, max_length=config['tokenizer']['encoder_max_len'], return_token_type_ids=False,)
test_tokenized_decoder_inputs = tokenizer(decoder_input_test, return_tensors="pt", padding=True,
                add_special_tokens=True, truncation=True, max_length=config['tokenizer']['decoder_max_len'], return_token_type_ids=False,)
```

4.2. Define `load_tokenizer_and_model_for_test`

: 추론을 위한 tokenizer와 학습시킨 모델을 checkpoint 경로를 통해 불러옵니다.

4.3. Define `inference`

: 학습된 모델이 생성한 요약문의 출력 결과를 보여줍니다.

```
********************************************************************** 대화문 **********************************************************************
 #Person1#: 더슨 씨, 받아쓰기 좀 해주세요.
#Person2#: 네, 실장님...
#Person1#: 이것은 오늘 오후까지 모든 직원에게 내부 메모로 전달되어야 합니다. 준비되셨나요?
#Person2#: 네, 실장님. 시작하셔도 됩니다.
#Person1#: 모든 직원들에게 주의하라... 즉시 효력을 발휘하여, 모든 사무실 통신은 이메일 통신과 공식 메모로 제한됩니다. 근무 시간 동안 직원들이 즉시 메시지 프로그램을 사용하는 것은 엄격히 금지됩니다.
#Person2#: 실장님, 이것은 내부 통신에만 적용되는 건가요? 아니면 외부 통신에도 제한이 되는 건가요?
#Person1#: 이것은 모든 통신에 적용되어야 합니다, 이 사무실 내의 직원들 사이뿐만 아니라 외부 통신에도 마찬가지입니다.
#Person2#: 하지만 실장님, 많은 직원들이 고객과 소통하기 위해 즉시 메시지를 사용하고 있습니다.
#Person1#: 그들은 그들의 의사소통 방법을 바꾸어야만 합니다. 이 사무실에서 누구도 즉시 메시지를 사용하지 않기를 원합니다. 너무 많은 시간을 낭비하게 됩니다! 이제, 메모를 계속해주세요. 우리가 어디까지 했나요?
#Person2#: 이것은 내부와 외부 통신에 적용됩니다.
#Person1#: 그렇습니다. 즉시 메시지를 계속 사용하는 어떤 직원이라도 먼저 경고를 받고 직무 정지에 처해질 것입니다. 두 번째 위반 시에는 직원은 해고에 처해질 것입니다. 이 새로운 정책에 대한 어떤 질문이라도 부서장에게 직접 문의하면 됩니다.
#Person2#: 그게 다신가요?
#Person1#: 네. 이 메모를 오후 4시 전에 모든 직원에게 타이핑하여 배포해 주세요.

********************************************************************** 요약문 **********************************************************************
   더슨 씨는 #Person1# 에게 모든 사무실 통신이 이메일 통신과 공식 메모로 제한된다고 알려준다. #Person2# 는 이것이 내부와 외부 통신에 적용된다고 설명한다.
```

코드 설명 (solar_api.ipynb)

1. Solar Chat API 요약 성능 확인하기

- Solar Chat API을 이용하여 train 및 validation dataset에 포함된 dialogue 샘플을 요약해 봅니다.

1.1 Define `compute_metrics`

: 모델 성능에 대한 평가 지표를 정의합니다. 본 대회에서는 ROUGE 점수를 통해 모델의 성능을 평가합니다.

1.2 Define `build_prompt`

: Dialogue를 입력으로 받아, Solar Chat API에 보낼 Prompt를 생성하는 함수를 정의합니다.

1.3 Define `summarization`

: Solar Chat API를 활용해 Summarization을 수행하는 함수를 정의합니다.

1.4 Define `test_on_train_data`

: Train data 중 처음 3개의 대화를 요약합니다. `num_samples` 값을 변경하여 원하는 양의 대화를 요약할 수 있습니다.
: For문을 이용하여 루프를 돌면서, 각각의 sample을 앞서 정의한 `summarization` 함수로 요약하고, 그 결과를 출력합니다.

```
Dialogue:
#Person1#: 안녕하세요, 스미스씨. 저는 호킨스 의사입니다. 오늘 왜 오셨나요?
#Person2#: 건강검진을 받는 것이 좋을 것 같아서요.
#Person1#: 그렇군요, 당신은 5년 동안 건강검진을 받지 않았습니다. 매년 받아야 합니다.
#Person2#: 알고 있습니다. 하지만 아무 문제가 없다면 왜 의사를 만나러 가야 하나요?
#Person1#: 심각한 질병을 피하는 가장 좋은 방법은 이를 조기에 발견하는 것입니다. 그러니 당신의 건강을 위해 최소한 매년 한 번은 오세요.
#Person2#: 알겠습니다.
#Person1#: 여기 보세요. 당신의 눈과 귀는 괜찮아 보입니다. 깊게 숨을 들이쉬세요. 스미스씨, 담배 피우시나요?
#Person2#: 네.
#Person1#: 당신도 알다시피, 담배는 폐암과 심장병의 주요 원인입니다. 정말로 끊으셔야 합니다.
#Person2#: 수백 번 시도했지만, 습관을 버리는 것이 어렵습니다.
#Person1#: 우리는 도움이 될 수 있는 수업과 약물들을 제공하고 있습니다. 나가기 전에 더 많은 정보를 드리겠습니다.
#Person2#: 알겠습니다, 감사합니다, 의사선생님.

Pred Summary: 호킨스 의사는 스미스씨에게 매년 건강검진을 받는 것이 심각한 질병을 조기에 발견하여 예방하는 데 중요하다고 조언합니다. 의사는 스미스씨의 눈과 귀를 검사하고, 폐암과 심장병의 위험을 증가시키는 흡연 습관을 끊을 것을 권장합니다. 의사는 스미스씨에게 도움을 줄 수 있는 수업과 약물에 대한 정보를 제공합니다.

Gold Summary: 스미스씨가 건강검진을 받고 있고, 호킨스 의사는 매년 건강검진을 받는 것을 권장합니다. 호킨스 의사는 스미스씨가 담배를 끊는 데 도움이 될 수 있는 수업과 약물에 대한 정보를 제공할 것입니다.

==========================================================================================
```

1.5 Define `validation`

: Validation data의 대화를 요약합니다. `num_samples` 값을 변경하여 원하는 양의 대화를 요약할 수 있습니다.
: For문을 이용하여 루프를 돌면서, 각각의 sample을 앞서 정의한 `summarization` 함수로 요약하고, `compute_metrics` 함수로 점수를 측정합니다. 이후, 최종 평균 점수를 출력합니다.

2. Solar Chat API로 요약하기

- Solar Chat API을 이용하여 test dataset에 포함된 dialogue를 요약하고 제출용 파일을 생성합니다.

2.1 Define `inference`

: Test data의 대화를 요약하여, 최종 제출 파일을 생성합니다.
: 요약 진행 시, RPM 제한에 걸리지 않도록, 분당 최대 100개의 요청만 하는 로직이 구현되어 있습니다.
: 요약이 완료되면 최종 결과 파일을 csv 형태로 저장합니다.

3. Prompt Engineering

- Prompt engineering을 통해 요약 성능 향상을 시도합니다.

2.1 Re-define `build_prompt`

: 앞서 정의한 `build_prompt` 함수를 재정의 하여, 다른 형태로 프롬프트를 생성하도록 합니다. 이후 기존 `test_on_train_data` 함수, `validation` 함수 및 `inference` 함수를 동일하게 이용하여 결과 확인, validation 진행 및 최종 추론 진행을 할 수 있습니다.

학습시간 및 학습장비

Stages GPU 서버를 활용했을 때의 실험 결과는 아래와 같습니다.

- Training time

  `CPU times: user 18min 10s, sys: 2min 20s, total: 20min 11s Wall time: 20min 56s`

- Inference time

  `CPU times: user 14.4 s, sys: 740 ms, total: 15.1 s Wall time: 12.6 s`

Public data 성능

제공된 baseline 기준 전체 test 데이터 중 250개의 public data에서 측정한 성능입니다.

- `ROUGE-F1 score: 47.1244`