Sentencepiece, kenlm

설치하기

- sentencepiece 설치

```
pip install sentencepiece
```

- kenlm 설치 (리눅스 / 맥OS / WSL)

```
sudo apt-get update
sudo apt-get install -y build-essential cmake libboost-all-dev zlib1g-dev

git clone https://github.com/kpu/kenlm.git
cd kenlm
mkdir build && cd build
cmake ..
make -j4
```

워크 플로우

1. 데이터 준비 – Python 개발 환경

- https://github.com/e9t/nsmc 에서 ratings_train.txt 와 ratings_test.txt를 다운로드
- panda를 사용하여 id와 label을 제거하고 내용만 txt파일로 각각 저장

2. SentencePiece BPE 기반 토크나이저 만들기 – Python 개발 환경

- SentencePiece를 이용하여 ratings_train.txt 파일을 대상으로 BPE 기반 model을 만듦
- 여기서 나오는 산출물은 model과 vocab 파일

3. KenLM 모델을 위한 토큰화된 텍스트 파일 만들기 – Python 개발 환경

- ratings_train.txt 파일을 BPE 모델에 입력하여 토큰화된 corpus.txt 파일을 생성

4. KenLM 모델 만들기 – Linux 콘솔 환경

- kenlm의 lmplz 명령을 이용하여 language model 생성 – arpa 파일 생성(텍스트)
- kenlm의 build_binary 명령을 이용하여 arpa lm을 bin lm으로 변환

5. BPE model과 kenlm binary model을 이용하여 ratings_text.txt 파일 대상으로 perplexity 시험 실시

1. 데이터 준비

데이터 셋 요구 조건

- 라인 하나에 하나의 문장이 있어야 성능 보장
- 내용은 연관성 있는 문장들로 구성되어 있는 것이 더 좋음
- 데이터 크기는 100K ~ 1M 문장의 적당한 크기로 하는 것이 실험에 적합

네이버 영화 리뷰 감성 분석 (Naver Sentiment Movie Corpus)

- 네이버 영화 리뷰를 기반으로 한 감성 분석 데이터셋
- 긍정/부정 레이블 있으나 SentencePiece나 KenLM 실험에는 레이블을 제외하고 텍스트 부분만 사용
- 약 20만 개의 리뷰로 구성되어 있어, 기본적인 실험 및 성능 평가에는 충분
- 일상적인 한국어 문장으로 이루어져 있어 범용성이 높음
- 코퍼스가 깔끔하게 정돈되어 있어 별도의 전 처리 과정이 많이 필요하지 않음

1. 데이터 준비

- 다음 사이트를 git clone해서 데이터 다운로드: https://github.com/e9t/nsmc
- 데이터 포맷은 csv 파일 포맷으로 되어있어서 panda를 사용하면 간단하게 텍스트만 추출할 수 있음

```
import pandas as pd


# ratings_train.txt 파일 불러오기
train_df = pd.read_csv('ratings_train.txt', sep='\t')


# document 컬럼의 내용만 추출하여 새 파일에 저장
with open('nsmc_train.txt', 'w', encoding='utf-8') as f:
    for text in train_df['document']:
        # NaN 값이 있을 수 있으므로 체크하고, 한 줄씩 쓰기
        if pd.notna(text):
            f.write(text + '\n')
```

2. SentencePiece BPE 기반 Tokenizer 만들기

옵션	설명	비고
--input=../data/nsmc_train.txt	학습에 사용할 입력 텍스트 파일	한 줄 = 한 문장
--model_prefix=spm_bpe32k	출력 모델 파일 이름 접두사	spm_bpe32k.model, spm_bpe32k.vocab 생성
--vocab_size=32000	어휘 크기 설정	32k 토큰
--model_type=bpe	모델 유형 선택	bpe, unigram, char, word 가능
--normalization_rule_name=nmt_nfkc	텍스트 정규화 규칙	유니코드 NFKC 정규화
--character_coverage=0.9995	문자 집합 커버율	rare char 무시 (99.95% 포함)
--byte_fallback=true	미등록 문자 처리	바이트 단위 토큰으로 대체
--hard_vocab_limit=false	어휘 크기 강제 여부	`true`면 정확히 vocab_size 맞춤
--input_sentence_size=2000000	전체 중 샘플링할 문장 개수	데이터가 클 때 사용
--shuffle_input_sentence=true	입력 문장 섞기	랜덤 샘플링 다양성 확보

2. SentencePiece 학습 – BPE

```
args = [
  '--input=../data/nsmc_train.txt',
  '--model_prefix=spm_bpe32k',
  '--vocab_size=32000',
  '--model_type=bpe',
  '--normalization_rule_name=nmt_nfkc',
  '--character_coverage=0.9995',
  '--byte_fallback=true',
  '--hard_vocab_limit=false',
  #'--input_sentence_size=2000000','--shuffle_input_sentence=true’  # 데이터 셋이 클 경우
]
spm.SentencePieceTrainer.Train(' '.join(args))

sp = spm.SentencePieceProcessor(model_file='nsmc_bpe32k.model')
strs = sp.encode('안녕하세요. 반갑습니다 🙂.', out_type=str)
print(strs)
ids = sp.encode('안녕하세요. 반갑습니다 🙂.', out_type=int)
print(ids)
org_str = sp.decode(ids)
print(org_str)
```

3. KenLM 모델을 위한 토큰화된 텍스트 파일 만들기

- 아 더빙.. 진짜 짜증나네요 목소리
- 흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나
- 너무재밋었다그래서보는것을추천한다
- 교도소 이야기구먼 ..솔직히 재미는 없다..평점 조정
- 사이몬페그의 익살스런 연기가 돋보였던 영화!스파이더맨에서 늙어보이기만 했던 커스틴 던스트가 너무나도 이뻐보였다

- SentencePiece로 만든 tokenizer를 이용하여 원본 텍스트를 토큰화된 텍스트로 변환하고 이를 KenLM 모델에 입력

- _아 _더빙 .. _진짜 _짜증나네요 _목소리
- _흠 ... 포스터 보고 _초딩영화 줄 .... 오버 연기 조차 _가볍지 _않 구나
- _너무 재 밋 었다 그래서 보는 것을 추천 한다
- _교도소 _이야기 구먼 _... 솔직히 _재미는 _없다 .. 평점 _조정
- _사이 몬 페 그의 _익살 스런 _연기가 _돋보였던 _영화 ! 스파이더맨 에서 _늙어 보이 기만 _했던 _커 스틴 _던 스트 가 _너무나도 _이뻐 보였다

4. KenLM 모델 만들기

```
## 1. 학습(ARPA 생성)
kenlm/build/bin/lmplz -o n_gram_차수_예_3 < train.txt > model_name.arpa
## 2. 바이너리 변환(빠른 로딩/쿼리)
kenlm/build/bin/build_binary model_name.arpa model_name.bin
## 3. 문장 점수 확인(테스트)
kenlm/build/bin/query model_name.bin < test.txt
## 4. Perplexity 확인
kenlm/build/bin/query -p model_name.bin < text.txt
### 자주 쓰는 옵션
학습: lmplz -o <N> (예: -o 5)
변환: build_binary <in_name.arpa> <out_name.bin>
질의: query <model_name.bin> (+ -p로 perplexity 출력되는 빌드가 많음)
```

4. KenLM 모델 만들기 – 주요 옵션

kenlm/build/bin/lmplz 주요 옵션

① -o <order> : n-gram 차수(order) 지정. (예) –o 3 → trigram model

② --text <file> : 입력 텍스트 파일 (보통 < file redirection 입력) (예) lmplz -o 3 < corpus.txt

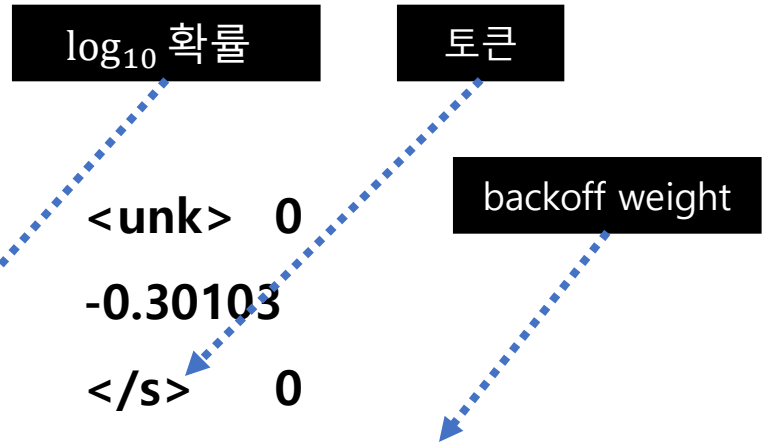
③ --arpa <file> : 출력 파일명 (ARPA 포맷) (예) lmplz -o 3 < corpus.txt > model_name.arpa

④ --discount_fallback : Kneser–Ney discount 값 부족 시 Katz backoff 사용(데이터 적을 때 유용)

⑤ --prune <threshold> : 특정 빈도 이하 n-gram 제거 (예) --prune 0 1 2 → unigram 그대로, bigram 빈도 1 이하 제거, trigram 빈도 2 이하 제거 (예) --prune 0 1 1 → unigram 그대로, bigram 빈도 1 이하 제거, trigram 빈도 1 이하 제거

사용 예: kenlm/build/bin/lmplz -o 5 --prune 0 1 2 < corpus.txt > model.arpa

4. KenLM 모델 만들기 – ARPA 파일 구조




```
\data\
ngram 1=19
ngram 2=20
ngram 3=20

\1-grams:
-1.5563025      <unk>   0
0       <s>     -0.30103
-1.2775489      </s>    0
-1.2775489      이      -0.30103
-1.2775489      모델은  -0.30103
-1.2775489      간단한  -0.30103

\2-grams:
-0.2786933      . </s>  0
-0.44403273     <s> 이  -0.30103
-0.5584794      이 모델은       -0.30103
-0.2786933      모델은 간단한   -0.30103

\3-grams:
-0.11736481     입니다 . </s>
-0.11736481     합니다 . </s>
-0.11736481     데이터입니다 . </s>
-0.4109507      <s> 이 모델은

\end\
```

5. KenLM 모델 이용하여 Perplexity 구하기 – score

```
lm.score(sentence: str, bos: bool = True, eos: bool = True) -> float
```

1. sentence: str
   - 공백으로 구분된 토큰 문자열
   - (예) "_아 _더빙 .. _진짜 _짜증나네요 _목소리".
   - 따라서, 일반 텍스트를 tokenizer로 encode해서 입력하여야 함.
2. bos: bool = True
   - BOS (begin-of-sentence) 토큰 <s> 포함 여부.
   - bos=True (기본값) → 문장 앞에 <s> 추가해서 확률 계산. (예) 실제 계산은 <s> 나는 학교에 간다 .
   - bos=False → 문장 시작 토큰 없이 계산.예: 나는 학교에 간다 . (예) 만 계산 (부분 시퀀스 평가에 사용).
3. eos: bool = True
   - EOS (end-of-sentence) 토큰 </s> 포함 여부.
   - eos=True (기본값) → 문장 끝에 </s> 추가해서 계산. (예) 나는 학교에 간다 . </s>
   - eos=False → 끝 토큰 제외. 아직 문장이 끝나지 않은 중간 토큰 시퀀스를 평가할 때 사용.
4. 출력: float
   - 내부적으로 n-gram 체인 규칙을 따라 확률을 곱해서 자연로그(ln) 값으로 반환

5. KenLM 모델 이용하여 Perplexity 구하기 – full_scores

- `lm.full_scores(sentence: str, bos: bool = True, eos: bool = True)`

1. 입력: sentence, bos, eos 는 'score'와 같다.
2. 출력: 토큰 별 스코어(iterable of tuples)

- 각 토큰 위치마다 (logprob, ngram_length, is_oov) 튜플 반환
- logprob: 해당 단어의 조건부 확률(자연로그, ln)
- ngram_length: 해당 단어를 예측할 때 사용된 n-gram 길이
- is_oov (out-of-vocabulary): 1 (True) 해당 단어가 ARPA에 없음, 0 (False) 정상 단어
- 예시 출력

```
Token: _지루          LogProb:   -3.42 NgramLen: 2 OOV: False
Token: 하지는         LogProb:   -1.62 NgramLen: 3 OOV: False
Token: _않은데        LogProb:   -2.32 NgramLen: 2 OOV: False
Token: _완전          LogProb:   -3.32 NgramLen: 1 OOV: False
Token: _막장임        LogProb:   -3.92 NgramLen: 2 OOV: False
```

5. KenLM 모델 이용하여 Perplexity 구하기

- lm.score 의 출력은 입력 문장이 나올 확률의 자연 로그이므로 perplexity는 다음과 같이 구할 수 있다.

$$PPL = exp\left(-\frac{logprob}{N}\right)$$

- `logprob = lm.score(sentence)`
- `N = len(sentence)`

- 예제
- 문장 단위 점수 계산
- 파일 단위 점수 계산
- 다음 토큰 후보 랭킹 계산