카페가 궁금해!
: 카페 리뷰 감성 분석

숙탯 딥러닝 프로젝트 1조
김희진 박소연 지상은 최윤서

목차

주제 선정 배경

너무 많은 카페 리뷰

모든 리뷰를 다 확인하지 않고도
가장 **긍정/부정**적인 것만 확인하면
가고싶은 카페의 장단점을
모두 파악할 수 있지 않을까?

카페 리뷰 **감성분석**을 통해
긍정/부정을 분류하는 모델을 만들고,
원하는 카페의 장단점을
한 눈에 볼 수 있는 프로그램을
구현해보자!

데이터 크롤링

카카오맵 카페 리뷰

가산동 논현동 명동 삼성동
서교동 서초동 성수동 압구정동
여의도동 역삼동 종로1/2/3/4가동

데이터 크롤링

카페이름, 리뷰내용, 사용자 리뷰 별점,
사용자 리뷰 개수, 사용 평균 별점

데이터 크롤링

```
df_crawling
```

	name	review	star	review_num	ave_star
0	인크커피 가산 플래그십 스토어	대형 카페 그잡채빵들이 너무 맛있다 커피 독특한데 맛남더보기	4.0	400	4.4
1	인크커피 가산 플래그십 스토어	이 가격에 이 용량에 1.5샷? 싸장님 맘이라지만 ㅎㅎㅎ 한번 가본걸로 이만더보기	2.0	62	2.1
2	인크커피 가산 플래그십 스토어	트리보러 가는 곳넓은데 사람 많음빵은 다 맛있었는데, 오히려 커피는 아쉬움더보기	4.0	25	4.1
3	인크커피 가산 플래그십 스토어	동네라 자주 가는데 어쩔땐 친절하고 어쩔땐 불친절하고 알바바이알바인듯 ㅋㅋㅋ 근데 ...	4.0	90	3.4
4	인크커피 가산 플래그십 스토어	카페 분위기는 좋으나 디저트 준비하는 곳에서 일하시는 여자 알바가 너무 싸가지 없어...	1.0	1	1.0
...	...	...	...	...	...
174880	설빙 노량진점	노량진점 설빙 진짜별로네요 최악이에요 고객한테 대하는 전화 말투법이 상당히 거칠고 ...	1.0	2	1.0
174881	설빙 노량진점	배달 시켰는데 다른 지점보다 양이 많아서 놀람.더보기	5.0	29	4.2
174882	설빙 노량진점	맛잇음 다른 설빙 가게들보다 양도 많은거 같고더보기	4.0	10	1.9
174883	설빙 노량진점	체육관 2개와 포켓스탑 1개 잡힘.영등포 설빙은 시설도 빙수 퀄리티도 최악인데 여...	4.0	318	3.4
174884	설빙 노량진점	학생들을 위한 배려인건지는 모르겠으나 설빙 다른 지점보다 토핑이나 재료를 후하게 얹...	4.0	26	3.3

174885 rows × 5 columns

데이터 전처리

·중복 제거

·리뷰 뒤에 붙은 '더보기', '접기' 제거

·리뷰 없는 데이터 제거

·리뷰 내용에서 한글만 남기고 제거 (외국어, 이모티콘 등)

사용자 리뷰 별점과
사용자 평균 별점을 통한
긍정(0)/부정(1) 라벨링

사용자 평균 별점이
너무 낮은 사용자
(악성 리뷰러)
&
너무 짧은 리뷰
(ex. 자음 하나)

=>삭제하려다가
데이터 손실 최소화를 위해
보존

데이터 전처리

·중복 제거

·리뷰 뒤에 붙은 '더보기', '접기' 제거

·리뷰 없는 데이터 제거

·리뷰 내용에서 한글만 남기고 제거 (외국어, 이모티콘 등)

```
review_unique = review_all.drop_duplicates(['review'], keep ='first')
```

```
review_unique.review = review_unique.review.str.strip('더보기접기')
```

```
review_unique = review_unique.drop(2,axis=0)
```

```
#df_copy['review']
review_unique['review']  = review_unique['review'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣 ]","")
```

데이터 전처리

사용자 리뷰 별점과
사용자 평균 별점을 통한
긍정(0)/부정(1) 라벨링

1, 2점: 부정　　**4, 5점: 긍정**

3점

· |(사용자평균별점-사용자 리뷰 별점|≤1.5이면

해당 리뷰 삭제

· (사용자 리뷰 별점-사용자평균별점)>1.5이면

긍정

· (사용자 리뷰 별점-사용자평균별점)< -1.5이면

부정

데이터 전처리

사용자 리뷰 별점과
사용자 평균 별점을 통한
긍정(0)/부정(1) 라벨링

사용자 리뷰 별점==3점 이고
|(사용자평균별점-사용자 리뷰 별점|≤1.5인 리뷰 삭제

```
# drop review_star = 3 &  | review_star - id_star | < 1.5
final = final.drop(final[(final['star']== 3)&( 1.4 < final['id_star'])&(final['id_star'] <4.6)].index)
```

1, 2, 4, 5점 라벨링

```
# 리뷰 평점 1,2,4,5라벨링 (긍정 : 0 , 부정 : 1)
def sentiment(x):
    if x > 3 :
        return '0'
    elif x < 3 :
        return '1'
```

```
final['sentiment'] = final['star'].apply(lambda x: sentiment(x))
```

데이터 전처리

사용자 리뷰 별점과
사용자 평균 별점을 통한
긍정(0)/부정(1) 라벨링

3점 라벨링

```
# 리뷰 평점 =9 라벨링
def sentiment3(x):
    if x >  4.5 :
        return '1'
    elif x < 1.5 :
        return '0'
```

- (사용자 리뷰 별점-사용자평균별점)>1.5이면 긍정
- (사용자 리뷰 별점-사용자평균별점)< -1.5이면 부정

```
final[final.sentiment.isna()].id_star.apply(lambda x : sentiment3(x))
```

```
6       1
175     1
476     1
860     1
2208    1
2334    0
3260    1
3518    1
```

데이터 전처리

사용자 리뷰 별점과
사용자 평균 별점을 통한
긍정(0)/부정(1) 라벨링

총 **33,053**개의
최종 데이터셋

```
final['sentiment'].fillna(final[final.sentiment.isna()].id_star.apply(lambda x : sentiment3(x)),inplace=True)
```

```
df_labelling
```

	name	review	star	review_num	ave_star	label
0	인크커피 가산 플래그십 스토어	대형 카페 그잡채빵들이 너무 맛있다 커피 독특한데 맛남	4.0	400	4.4	0
1	인크커피 가산 플래그십 스토어	이 가격에 이 용량에 샷 싸장님 맘이라지만 ㅎㅎㅎ 한번 가본걸로 이만	2.0	62	2.1	1
2	인크커피 가산 플래그십 스토어	트리보러 가는 곳넓은데 사람 많음빵은 다 맛있었는데 오히려 커피는 아쉬움	4.0	25	4.1	0
3	인크커피 가산 플래그십 스토어	둥네라 자주 가는데 어쩔땐 친절하고 어쩔땐 불친절하고 알바바이알바인듯 ㅋㅋㅋ 근데 ...	4.0	90	3.4	0
4	인크커피 가산 플래그십 스토어	카페 분위기는 좋으나 디저트 준비하는 곳에서 일하시는 여자 알바가 너무 싸가지 없어...	1.0	1	1.0	1
...	...	...	...	...	...	...
33048	설빙 노량진점	노량진점 설빙 진짜별로네요 최악이에요 고객한테 대하는 전화 말투법이 상당히 거칠고 ...	1.0	2	1.0	1
33049	설빙 노량진점	배달 시켰는데 다른 지점보다 양이 많아서 놀람	5.0	29	4.2	0
33050	설빙 노량진점	맛잇음 다른 설빙 가게들보다 양도 많은거 같고	4.0	10	1.9	0
33051	설빙 노량진점	체육관 개와 포켓스탑 개 잡힘영등포 설빙은 시설도 빙수 퀄리티도 최악인데 여기는 ...	4.0	318	3.4	0
33052	설빙 노량진점	학생들을 위한 배려인건지는 모르겠으나 설빙 다른 지점보다 토핑이나 재료를 후하게 얹...	4.0	26	3.3	0

33053 rows × 6 columns

BERT모델

BERT(**B**idirectional **E**ncoder **R**epresentations from **T**ransformers)

학습 과정

: 토큰 임베딩,
세그먼트 임베딩,
포지셔닝 임베딩
→ 사전 훈련(MLM, NSP)
→ 전이 학습.

전처리

: 앞뒤에 토큰 넣기,
토크나이징, 패딩,
어텐션 마스크

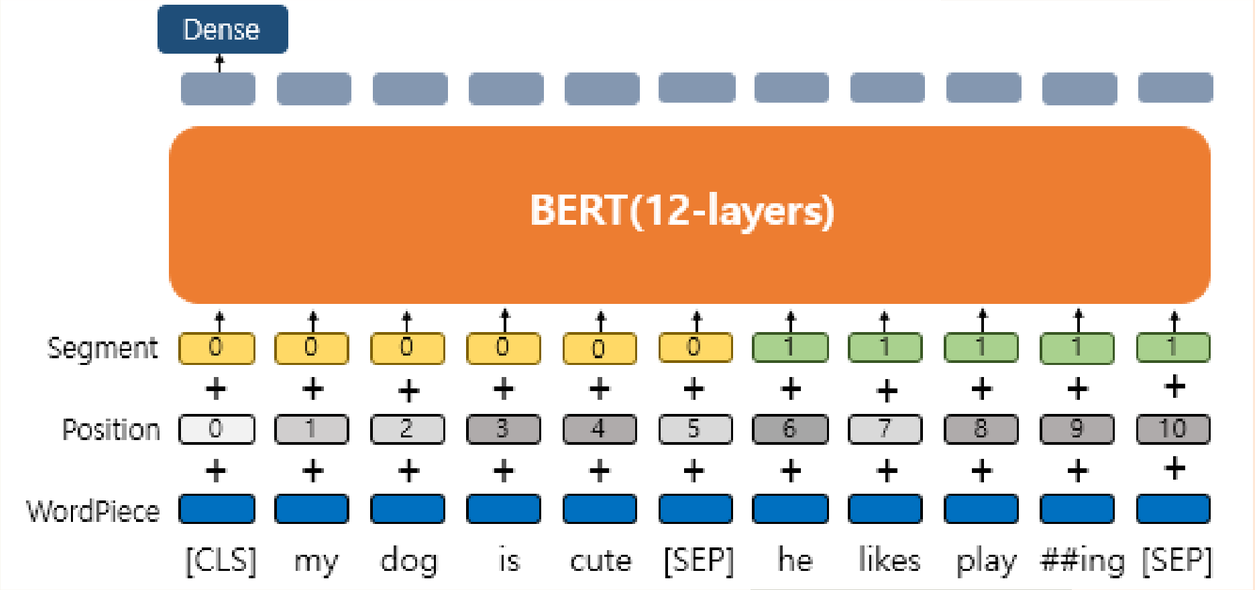
BERT모델

<내부 동작 과정>

·토큰 임베딩

·세그먼트 임베딩

·포지셔닝 임베딩



BERT모델

토큰 임베딩

•실질적인 입력이 되는 워드 임베딩
•문자 단위로 임베딩
•자주 등장하면서 가장 길이가 긴 sub-word를 하나의 단위로 분리

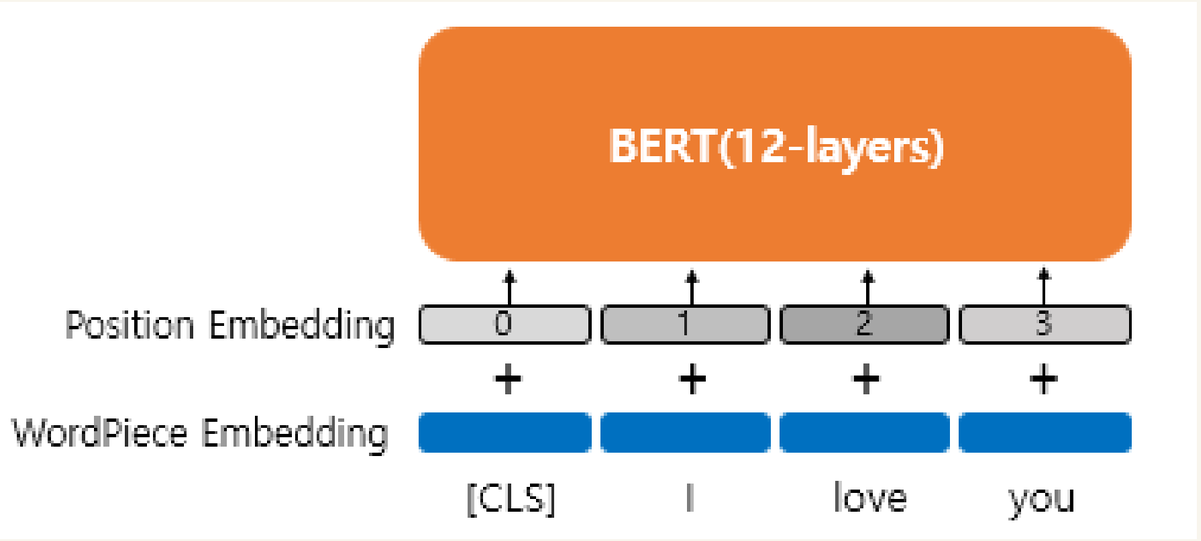
```
result = tokenizer.tokenize('Here is the sentence I want embeddings for.')
print(result)
```

['here', 'is', 'the', 'sentence', 'i', 'want', 'em', '##bed', '##ding', '##s', 'for', '.']

BERT모델

포지셔닝 임베딩

·위치 정보를 학습하기 위한 임베딩

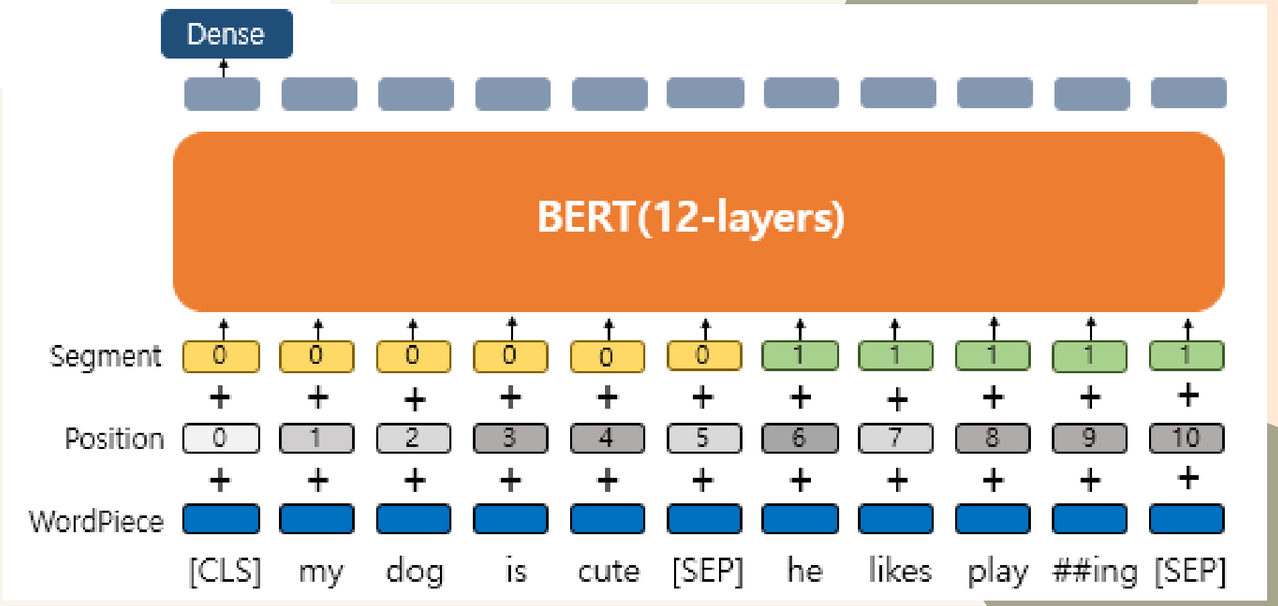


- 첫번째 단어의 임베딩 벡터 + 0번 포지션 임베딩 벡터
- 두번째 단어의 임베딩 벡터 + 1번 포지션 임베딩 벡터
- 세번째 단어의 임베딩 벡터 + 2번 포지션 임베딩 벡터
- 네번째 단어의 임베딩 벡터 + 3번 포지션 임베딩 벡터

BERT모델

세그먼트 임베딩

·두 개의 문장을 구분하기 위한 임베딩
·첫번째 문장에는 Sentence 0 임베딩,
두번째 문장에는 Sentence 1 임베딩을
더해주는 방식



BERT모델

<사전 훈련>

-MLM

·빈칸 뚫고 맞추기 학습

·입력 텍스트의 15% 단어를 랜덤으로 마스킹

·이후 신경망이 가려진 단어들을 예측하도록 함

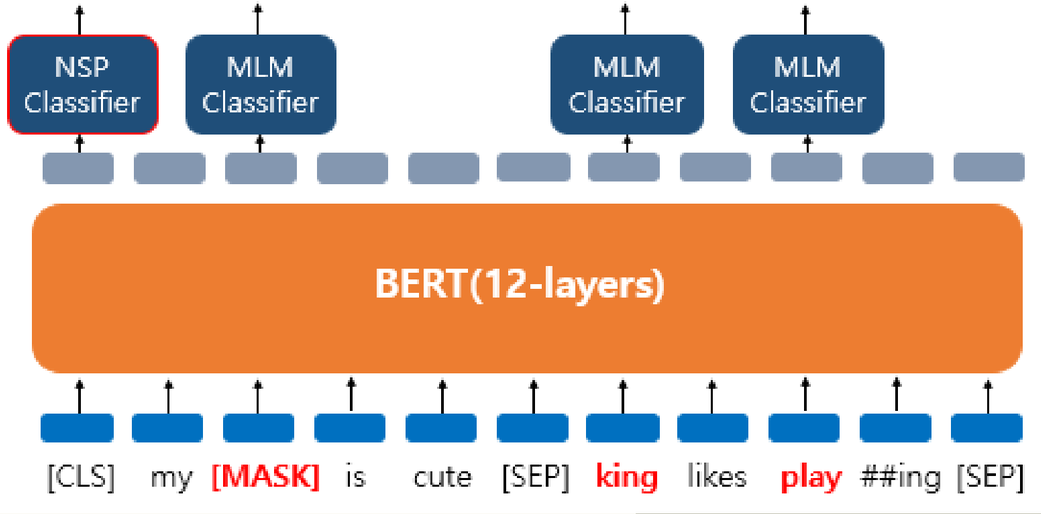
-NSP

·다음 문장 예측

·두 개의 문장을 준 후에

이 문장이 이어지는 문장인지 아닌지를

맞추는 방식으로 훈련



BERT모델

<전처리>

-구분자 넣기

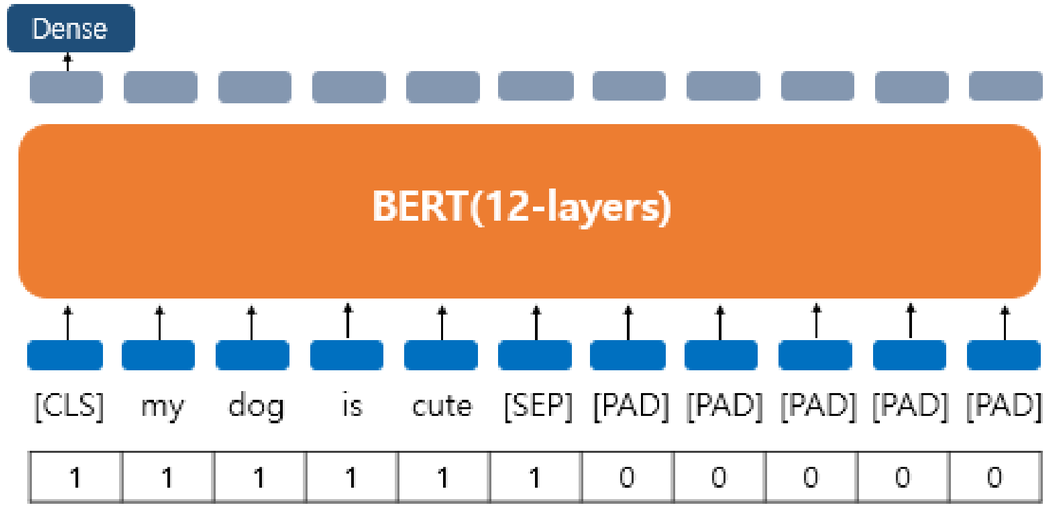
·각 문장의 앞마다 [CLS]구분자를,
뒤마다 [SEP]구분자를 붙임

-패딩

·설정한 MAX_LEN 만큼 빈 공간을 0으로 채움

-어텐션 마스크

·불필요하게 패딩 토큰에 대해서 어텐션을 하지 않도록
실제 단어와 패딩 토큰을 구분할 수 있도록 알려주는 입력
·숫자 1: 해당 토큰은 실제 단어
·숫자 0: 해당 토큰은 패딩 토큰



BERT모델

_사전준비

```
# Hugging Face의 트랜스포머 모델을 설치
pip install transformers
```

```
# 주요 패키지 불러오기
import torch

from transformers import BertTokenizer
from transformers import BertForSequenceClassification, AdamW, BertConfig
from transformers import get_linear_schedule_with_warmup

from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split

import pandas as pd
import numpy as np
import random
import time
import datetime
```

http://yonghee.io/bert_binary_classification_naver/

BERT모델

_사전준비

```
# 구글 드라이브 마운트
from google.colab import drive
drive.mount('/content/drive')
```

```
# GPU 확인
# torch에서 GPU가 사용되는지 확인하는 코드
# Colab 사용시에도 어떤 GPU를 쓰는지 알 수 있음.

import os

n_devices = torch.cuda.device_count()
print(n_devices)

for i in range(n_devices):
    print(torch.cuda.get_device_name(i))
```

```
1
Tesla T4
```

BERT모델

_데이터로드

```
# train / test 로드 8:2


train = pd.read_csv("/content/drive/MyDrive/학회/train.csv")
test = pd.read_csv("/content/drive/MyDrive/학회/test.csv")

##train = pd.read_csv("/Users/parksoyeon/Desktop/학회/딥러닝프로젝트/data/train_test/train.csv")
##test = pd.read_csv("/Users/parksoyeon/Desktop/학회/딥러닝프로젝트/data/train_test/test.csv")

print('train shape:', train.shape)
print('test shape:', test.shape)
```

```
train shape: (26442, 7)
test shape: (6611, 7)
```

- 훈련셋 26442개
- 테스트셋 6611개

- label

0: 긍정 1 : 부정

BERT모델

_전처리

```
# BERT의 입력 형식에 맞게 변환
# BERT 분류모델은 각 문장의 앞마다 [CLS]를 붙여 인식, 종료는 [SEP]
print('sentence processing..')

review_bert = ["[CLS]"+str(s)+"[SEP]" for s in train.review]
review_bert[:5]
```

```
sentence processing..
['[CLS]여기 여사장불친절에 조현병환잔줄 인성ㅆㄹㄱ에 기분 나빴는데 역시나였구나ㅋㅋㅋ 거르는 카페 점이없어서 아쉽다[SEP]',
 '[CLS]바질토마토에이드먹다 기영이 될거같음달콤하고 상큼하고 맛있어요주호님 팬은 아니고 친구 따라 간건데웨이팅 할 이유가 충분함 존맛[SEP]',
 '[CLS]망고코코넛스무디존맛 크루아상샌드위치미침[SEP]',
 '[CLS]너무귀엽고 직원분들 친절하세요ㅜㅜ[SEP]',
 '[CLS]개인적으로 여기 공차는 내가 가본데중 제일 맛있음[SEP]']
```

BERT모델

_전처리

```
# BERT의 입력 형식에 맞게 변환
# BERT 분류모델은 각 문장의 앞마다 [CLS]를 붙여 인식, 종료는 [SEP]
print('sentence processing..')

review_bert = ["[CLS]"+str(s)+"[SEP]" for s in train.review]
review_bert[:5]
```

```
sentence processing..
['[CLS]여기 여사장불친절에 조현병환잔줄 인성ㅆㅅㄹㄱ에 기분 나빴는데 역시나였구나ㅋㅋㅋ 거르는 카페 점이없어서 아쉽다[SEP]',
 '[CLS]마질토마토에이드먹다 기영이 될거같음달콤하고 상큼하고 맛있어요주호님 팬은 아니고 친구 따라 간건데웨이팅 할 [SEP]',
 '[CLS]망고코코넛스무디존맛 크루아상샌드위치[SEP]',
 '[CLS]너무귀엽고 직원분들 친절하세요ㅜㅜ[SEP]',
 '[CLS]개인적으로 여기 공차는 내가 가본데중 제일 맛있음[SEP]']
```

BERT모델

_토크나이징

```
print('tokenizeing..')

# BERT의 토크나이저로 문장을 토큰으로 분리
tokenizer = BertTokenizer.from_pretrained('bert-base-multilingual-cased',do_lower_case=False)
tokenized_texts = [ tokenizer.tokenize(s) for s in review_bert]

tokenized_texts[0]
```

```
tokenizeing..
Downloading (...)solve/main/vocab.txt: 100% 996k/996k [00:01<00:00, 857kB/s]
Downloading (...)okenizer_config.json: 100% 29.0/29.0 [00:00<00:00, 1.09kB/s]
Downloading (...)lve/main/config.json: 100% 625/625 [00:00<00:00, 27.9kB/s]
```

```
['[CLS]',
 '여',
 '##기',
 '여',
 '##사',
 '##장',
 '##불',
 '##친',
 '##절',
 '##에',
 '조',
 '##현',
 '##병',
 '##환',
 '##잔',
 '##줄',
 '[UNK]',
 '기',
 '##분',
 '[UNK]',
 '[UNK]',
 '거',
 '##르는',
 '카',
 '##페',
 '점',
 '##이',
 '##없',
 '##어',
 '##서',
 '아',
 '##쉽',
 '##다',
 '[SEP]']
```

BERT모델

_패딩

```
print('padding')

# 입력 토큰의 최대 시퀀스 길이
MAX_LEN = 128

# 토큰을 숫자 인덱스로 변환
input_ids = [tokenizer.convert_tokens_to_ids(x) for x in tokenized_texts]

# 문장을 MAX_LEN 길이에 맞게 자르고, 모자란 부분을 패딩 0으로 채움
input_ids = pad_sequences(input_ids, maxlen = MAX_LEN, dtype='long', truncating='post', padding='post')

input_ids[0]
```

```
padding
array([   101,   9565,  12310,   9565,  12945,  13890, 118992,  55358,
        58931,  10530,   9678,  30842,  73380,  51745, 119196, 119219,
          100,   8932,  37712,    100,    100,   8863,  47908,   9786,
       119391,   9668,  10739, 119136,  12965,  12424,   9519, 119072,
        11903,    102,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0])
```

BERT모델

_패딩

```
print('padding')

# 입력 토큰의 최대 시퀀스 길이
MAX_LEN = 128

# 토큰을 숫자 인덱스로 변환
input_ids = [tokenizer.convert_tokens_to_ids(x) for x in tokenized_texts]

# 문장을 MAX_LEN 길이에 맞게 자르고, 모자란 부분을 패딩 0으로 채움
input_ids = pad_sequences(input_ids, maxlen = MAX_LEN, dtype='long', truncating='post', padding='post')

input_ids[0]
```

```
padding
array([   101,   9565,  12310,   9565,  12945,  13890, 118992,  55358,
        58931,  10530,   9678,  30842,  73380,  51745, 119196, 119219,
          100,   8932,  37712,    100,    100,   8863,  47908,   9786,
       119391,   9668,  10739, 119136,  12965,  12424,   9519, 119072,
        11903,    102,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0])
```

패딩된 부분

BERT모델

_어텐션마스크

```
#학습 속도를 높이기 위해 실 데이터가 있는 곳과 padding이 있는곳을 attention에게 알려줌


# 어텐션 마스크 초기화
attention_masks = []

# 어텐션 마스크를 패딩이 아니면 1, 패딩이면 0으로 설정
# 패딩 부분은 BERT 모델에서 어텐션을 수행하지 않아 속도 향상
for seq in input_ids:
    seq_mask = [float(i>0) for i in seq]
    attention_masks.append(seq_mask)

attention_masks[0]
```

```
[1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 0.0,
 0.0,
 0.0,
```

BERT모델

_어텐션마스크

```
#학습 속도를 높이기 위해 실 데이터가 있는 곳과 padding이 있는곳을 attention에게 알려줌


# 어텐션 마스크 초기화
attention_masks = []

# 어텐션 마스크를 패딩이 아니면 1, 패딩이면 0으로 설정
# 패딩 부분은 BERT 모델에서 어텐션을 수행하지 않아 속도 향상
for seq in input_ids:
    seq_mask = [float(i>0) for i in seq]
    attention_masks.append(seq_mask)

attention_masks[0]
```

어텐션 마스크 패딩 x

어텐션 마스크 패딩 o

BERT모델

_Train, Validation 분리

```
# input 과 mask가 뒤섞이지 않도록 random_state를 일정하게 고정.

# test set은 위에서 분리되었기에 , train 과 validation set만 분리
print('split train - val')

# 훈련셋과 검증셋으로 분리
train_inputs, validation_inputs, train_labels, validation_labels = ₩
train_test_split(input_ids, train['label'].values, random_state=42, test_size=0.1)

# 어텐션 마스크를 훈련셋과 검증셋으로 분리
train_masks, validation_masks, _, _ = train_test_split(attention_masks, input_ids, random_state=42,test_size=0.1)
```

```
split train - val
```

BERT모델

_파이토치 텐서로 변환

```
# numpy ndarray로 되어있는 input,lable,mask들을 torch tensor로 변환

# 데이터를 파이토치의 텐서로 변환
print('convert data to tenser..')

train_inputs = torch.tensor(train_inputs)
train_labels = torch.tensor(train_labels)
train_masks = torch.tensor(train_masks)
validation_inputs = torch.tensor(validation_inputs)
validation_labels = torch.tensor(validation_labels)
validation_masks = torch.tensor(validation_masks)
```

```
convert data to tenser..
```

BERT모델

_배치 및 데이터로더 설정

```
#현재 쓰고있는 GPU의 VRAM에 맞게 배치사이즈 설정(크게 설정후 부족메시지가 뜨면 8의 배수중 작은것으로 줄여나가기)

# 배치 사이즈
BATCH_SIZE = 32

# 파이토치의 DataLoader로 입력, 마스크, 라벨을 묶어 데이터 설정
# 학습시 배치 사이즈 만큼 데이터를 가져옴
train_data = TensorDataset(train_inputs, train_masks, train_labels)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler = train_sampler, batch_size = BATCH_SIZE)

validation_data = TensorDataset(validation_inputs, validation_masks, validation_labels)
validation_sampler = SequentialSampler(validation_data)
validation_dataloader = DataLoader(validation_data, sampler=validation_sampler, batch_size=BATCH_SIZE)
```

set batch and data loader

BERT모델

_테스트셋 전처리

```
#위의 train-val 셋 전처리와 동일

print('data split')
sentences = test['review']
sentences = ["[CLS]"+str(sentence)+"[SEP]" for sentence in sentences]
labels = test['label'].values

tokenized_texts = [tokenizer.tokenize(sent) for sent in sentences]

input_ids = [tokenizer.convert_tokens_to_ids(x) for x in tokenized_texts]
input_ids = pad_sequences(input_ids, maxlen=MAX_LEN, dtype="long", truncating="post", padding="post")

attention_masks = []
for seq in input_ids:
        seq_mask = [float(i>0) for i in seq]
        attention_masks.append(seq_mask)

test_inputs = torch.tensor(input_ids)
test_labels = torch.tensor(labels)
test_masks = torch.tensor(attention_masks)

test_data = TensorDataset(test_inputs, test_masks, test_labels)
test_sampler = RandomSampler(test_data)
test_dataloader = DataLoader(test_data, sampler=test_sampler, batch_size=BATCH_SIZE)
```

```
data split
```

BERT모델

_모델 학습

```
#GPU 체크 및 할당
if torch.cuda.is_available():
    device = torch.device("cuda")
    print('There are %d GPU(s) available.' % torch.cuda.device_count())
    print('We will use the GPU:', torch.cuda.get_device_name(0))
else:
    device = torch.device("cpu")
    print('No GPU available, using the CPU instead.')

## 분류를 위한 BERT 모델 생성
# transformers 의 BertForSequenceClassification 모듈 이용
# 이진분류 이므로 num_labels는 2로 설정
print('making BERT model for classification')
model = BertForSequenceClassification.from_pretrained("bert-base-multilingual-cased", num_labels=2)
#model.cuda()
model.to(device)
```

BERT모델

_학습 스케쥴링

```
# transformers에서 제공하는 옵티마이저 중 AdamW를 사용
# 총 훈련 스텝은 이터레이션 * 에폭 수로 설정
# 러닝 레잇 스케쥴러도 transformers에서 제공하는것을 사용
print('schedule start')

#옵티마이저 설정
optimizer = AdamW(model.parameters(),
                  lr = 2e-5, # 학습률
                  eps = 1e-8 # 0으로 나누는 것을 방지하기 위한 epsilon 값
                )

# 에폭수
epochs = 4

# 총 훈련 스텝 : 배치반복 횟수 * 에폭
total_steps = len(train_dataloader) * epochs

# 처음에 학습률을 lr 조금씩 감소시키는 스케줄러 생성
scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps = 0,
                                            num_training_steps = total_steps)
```

```
##학습

# accuracy 와 시간 표시함수 정의

# 정확도 계산 함수
print('train start')
def flat_accuracy(preds, labels):
    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    return np.sum(pred_flat == labels_flat) / len(labels_flat)

# 시간 표시 함수
def format_time(elapsed):
    # 반올림
    elapsed_rounded = int(round((elapsed)))
    # hh:mm:ss으로 형태 변경
    return str(datetime.timedelta(seconds=elapsed_rounded))
```

BERT모델

_학습 실행

```
# 데이터로더에서 배치만큼 가져온 후 forward, backward pass를 수행
# gradient update는 명시적으로 하지 않고 위에서 로드한 optimizer를 활용
# 재현을 위해 랜덤시드 고정
# 모든 Epoch를 학습하면 학습이 종료

seed_val = 42
random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)

# 그래디언트 초기화
model.zero_grad()
```

BERT모델

_학습 실행

```
# 에폭만큼 반복
for epoch_i in range(0, epochs):

    # ========================================
    #               Training
    # ========================================

    print("")
    print('======== Epoch {:} / {:} ========'.format(epoch_i + 1, epochs))
    print('Training...')

    # 시작 시간 설정
    t0 = time.time()

    # 로스 초기화
    total_loss = 0

    # 훈련모드로 변경
    model.train()
```

```
# 데이터로더에서 배치만큼 반복하여 가져옴
for step, batch in enumerate(train_dataloader):
    # 경과 정보 표시
    if step % 500 == 0 and not step == 0:
        elapsed = format_time(time.time() - t0)
        print('  Batch {:>5,}  of  {:>5,}.    Elapsed: {:}.'.format(step, len(train_dataloader), elapsed))

    # 배치를 GPU에 넣음
    batch = tuple(t.to(device) for t in batch)

    # 배치에서 데이터 추출
    b_input_ids, b_input_mask, b_labels = batch

    # Forward 수행
    outputs = model(b_input_ids,
                    token_type_ids=None,
                    attention_mask=b_input_mask,
                    labels=b_labels)

    # 로스 구함
    loss = outputs[0]

    # 총 로스 계산
    total_loss += loss.item()

    # Backward 수행으로 그래디언트 계산
    loss.backward()

    # 그래디언트 클리핑
    torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

    # 그래디언트를 통해 가중치 파라미터 업데이트
    optimizer.step()

    # 스케줄러로 학습률 감소
    scheduler.step()

    # 그래디언트 초기화
    model.zero_grad()
```

BERT모델

_학습 실행

```
# 평균 로스 계산
avg_train_loss = total_loss / len(train_dataloader)

print("")
print("  Average training loss: {0:.2f}".format(avg_train_loss))
print("  Training epcoh took: {:}".format(format_time(time.time() - t0)))

# ========================================
#               Validation
# ========================================

print("")
print("Running Validation...")

#시작 시간 설정
t0 = time.time()

# 평가모드로 변경
model.eval()

# 변수 초기화
eval_loss, eval_accuracy = 0, 0
nb_eval_steps, nb_eval_examples = 0, 0
```

```
# 데이터로더에서 배치만큼 반복하여 가져옴
for batch in validation_dataloader:
    # 배치를 GPU에 넣음
    batch = tuple(t.to(device) for t in batch)

    # 배치에서 데이터 추출
    b_input_ids, b_input_mask, b_labels = batch

    # 그래디언트 계산 안함
    with torch.no_grad():
        # Forward 수행
        outputs = model(b_input_ids,
                        token_type_ids=None,
                        attention_mask=b_input_mask)

    # 로스 구함
    logits = outputs[0]

    # CPU로 데이터 이동
    logits = logits.detach().cpu().numpy()
    label_ids = b_labels.to('cpu').numpy()

    # 출력 로짓과 라벨을 비교하여 정확도 계산
    tmp_eval_accuracy = flat_accuracy(logits, label_ids)
    eval_accuracy += tmp_eval_accuracy
    nb_eval_steps += 1

print("  Accuracy: {0:.2f}".format(eval_accuracy/nb_eval_steps))
print("  Validation took: {:}".format(format_time(time.time() - t0)))

print("")
print("Training complete!")
print("")
```

BERT모델

_학습 실행

```
======== Epoch 1 / 4 ========
Training...
  Batch   500  of    744.    Elapsed: 0:05:29.

  Average training loss: 0.29
  Training epcoh took: 0:08:09

Running Validation...
  Accuracy: 0.92
  Validation took: 0:00:17

======== Epoch 2 / 4 ========
Training...
  Batch   500  of    744.    Elapsed: 0:05:27.

  Average training loss: 0.18
  Training epcoh took: 0:08:07

Running Validation...
  Accuracy: 0.92
  Validation took: 0:00:17
```

validation set 정확도

: 0.92~0.93

```
======== Epoch 3 / 4 ========
Training...
  Batch   500  of    744.    Elapsed: 0:05:27.

  Average training loss: 0.13
  Training epcoh took: 0:08:07

Running Validation...
  Accuracy: 0.93
  Validation took: 0:00:17

======== Epoch 4 / 4 ========
Training...
  Batch   500  of    744.    Elapsed: 0:05:27.

  Average training loss: 0.09
  Training epcoh took: 0:08:07

Running Validation...
  Accuracy: 0.93
  Validation took: 0:00:17

Training complete!
```

BERT모델

_테스트셋 평가

```
print('test start')
#시작 시간 설정
t0 = time.time()

# 평가모드로 변경
model.eval()

# 변수 초기화
eval_loss, eval_accuracy = 0, 0
nb_eval_steps, nb_eval_examples = 0, 0

# 데이터로더에서 배치만큼 반복하여 가져옴
for step, batch in enumerate(test_dataloader):
    # 경과 정보 표시
    if step % 100 == 0 and not step == 0:
        elapsed = format_time(time.time() - t0)
        print('  Batch {:>5,}  of  {:>5,}.    Elapsed: {:}.'.format(step, len(test_dataloader), elapsed))

    # 배치를 GPU에 넣음
    batch = tuple(t.to(device) for t in batch)

    # 배치에서 데이터 추출
    b_input_ids, b_input_mask, b_labels = batch

    # 그래디언트 계산 안함
    with torch.no_grad():
        # Forward 수행
        outputs = model(b_input_ids,
                        token_type_ids=None,
                        attention_mask=b_input_mask)

    # 로스 구함
    logits = outputs[0]

    # CPU로 데이터 이동
    logits = logits.detach().cpu().numpy()
    label_ids = b_labels.to('cpu').numpy()

    # 출력 로짓과 라벨을 비교하여 정확도 계산
    tmp_eval_accuracy = flat_accuracy(logits, label_ids)
    eval_accuracy += tmp_eval_accuracy
    nb_eval_steps += 1

print("")
print("Accuracy: {0:.2f}".format(eval_accuracy/nb_eval_steps))
print("Test took: {:}".format(format_time(time.time() - t0)))
print("test finished!")
```

test set 정확도

: 0.93

```
test start
  Batch   100  of    207.    Elapsed: 0:00:21.
  Batch   200  of    207.    Elapsed: 0:00:43.

Accuracy: 0.93
Test took: 0:00:44
test finished!
```

BERT모델

_KoBERT

BERT는 영어보다 한국어에 대해서
정확도가 떨어짐

한국어 위키 5백만 문장과
한국어 뉴스 2천만 문장을 학습하여
한국어 정확도를 높인 모델이 바로
KoBERT

https://velog.io/@seolini43/KOBERT%EB%A1%9C-%EB%8B%A4%EC%A4%91-%EB%B6%84%EB%A5%98-%EB%AA%A8%EB%8D%B8-%EB%A7%8C%EB%93%A4%EA%B8%B0-%ED%8C%8C%EC%9D%B4%EC%8D%ACColab

BERT모델

_KoBERT

토큰화

```
tokenizer = get_tokenizer()
tok = nlp.data.BERTSPTokenizer(tokenizer, vocab, lower=False)

data_train = BERTDataset(dataset_train, 0, 1, tok, max_len, True, False)
data_test = BERTDataset(dataset_test, 0, 1, tok, max_len, True, False)
```

```
using cached model. /content/.cache/kobert_news_wiki_ko_cased-1087f8699e.spiece
```

BERT모델

_KoBERT

토큰화 결과

패딩된 시퀀스

```
data_train[0]
```

```
(array([   2, 3301, 3298, 6493, 7178, 6424, 7489, 7217, 6896, 4162, 7903,
        6361, 7946, 7172, 7292, 3758, 6573,    0,  490, 6896, 1282, 1370,
           0, 5761, 3327, 5655, 6944, 5496,  492,  492,  492,  862, 6113,
        5760, 4641, 4081, 6881, 6855, 6553, 3093, 6657, 5782,    3,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1], dtype=int32),
 array(43, dtype=int32),
 array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       dtype=int32),
 1)
```

BERT모델

_KoBERT

토큰화 결과

```
data_train[0]
```

```
(array([   2, 3301, 3298, 6493, 7178, 6424, 7489, 7217, 6896, 4162, 7903,
        6361, 7946, 7172, 7292, 3758, 6573,    0,  490, 6896, 1282, 1370,
           0, 5761, 3327, 5655, 6944, 5496,  492,  492,  492,  862, 6113,
        5760, 4641, 4081, 6881, 6855, 6553, 3093, 6657, 5782,    3,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1], dtype=int32),
 array(43, dtype=int32),
 array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       dtype=int32),
 1)
```

길이와 타입에 대한 내용 (패딩 되지 않은 길이)

BERT모델

_KoBERT

토큰화 결과

```
data_train[0]
```

```
(array([   2, 3301, 3298, 6493, 7178, 6424, 7489, 7217, 6896, 4162, 7903,
        6361, 7946, 7172, 7292, 3758, 6573,    0,  490, 6896, 1282, 1370,
           0, 5761, 3327, 5655, 6944, 5496,  492,  492,  492,  862, 6113,
        5760, 4641, 4081, 6881, 6855, 6553, 3093, 6657, 5782,    3,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1], dtype=int32),
 array(43, dtype=int32),
 array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       dtype=int32),
 1)
```

어텐션 마스크 시퀀스

BERT모델

_KoBERT

토큰화 결과

```
data_train[0]
```

```
(array([   2, 3301, 3298, 6493, 7178, 6424, 7489, 7217, 6896, 4162, 7903,
        6361, 7946, 7172, 7292, 3758, 6573,    0,  490, 6896, 1282, 1370,
           0, 5761, 3327, 5655, 6944, 5496,  492,  492,  492,  862, 6113,
        5760, 4641, 4081, 6881, 6855, 6553, 3093, 6657, 5782,    3,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1], dtype=int32),
 array(43, dtype=int32),
 array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       dtype=int32),
 1)
```

긍정
부정
라벨

BERT모델

_KoBERT

```
<ipython-input-26-480b6a139979>:5: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for batch_id, (token_ids, valid_length, segment_ids, label) in enumerate(tqdm_notebook(train_dataloader))
100% 414/414 [05:08<00:00, 1.78it/s]
epoch 1 batch id 1 loss 0.7343316674232483 train acc 0.53125
epoch 1 batch id 201 loss 0.21704243123531342 train acc 0.8155317164179104
epoch 1 batch id 401 loss 0.14899942278862 train acc 0.8628428927680798
epoch 1 train acc 0.8646965579710145
<ipython-input-26-480b6a139979>:23: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for batch_id, (token_ids, valid_length, segment_ids, label) in enumerate(tqdm_notebook(test_dataloader)):
100% 104/104 [00:26<00:00, 3.96it/s]
epoch 1 test acc 0.9300828694331984
100% 414/414 [05:06<00:00, 1.78it/s]
epoch 2 batch id 1 loss 0.14488670229911804 train acc 0.953125
epoch 2 batch id 201 loss 0.11172285676002502 train acc 0.9266169154228856
epoch 2 batch id 401 loss 0.0533280074596405 train acc 0.938707917705735 7
epoch 2 train acc 0.9390851449275363
100% 104/104 [00:26<00:00, 3.97it/s]
epoch 2 test acc 0.9359422444331984
100% 414/414 [05:07<00:00, 1.79it/s]
epoch 3 batch id 1 loss 0.019421160221099854 train acc 1.0
epoch 3 batch id 201 loss 0.019938861951231956 train acc 0.9584110696517413
epoch 3 batch id 401 loss 0.005084637552499771 train acc 0.9661393391521197
epoch 3 train acc 0.9663345410628019
100% 104/104 [00:26<00:00, 3.95it/s]
epoch 3 test acc 0.9356417636639676
```

```
100% 104/104 [00:26<00:00, 3.95it/s]
epoch 3 test acc 0.9356417636639676
100% 414/414 [05:06<00:00, 1.77it/s]
epoch 4 batch id 1 loss 0.011693246662616 73 train acc 1.0
epoch 4 batch id 201 loss 0.007661153562366962 train acc 0.9773009950248757
epoch 4 batch id 401 loss 0.003129021264612 6747 train acc 0.9823488154613467
epoch 4 train acc 0.9822992149758454
100% 104/104 [00:26<00:00, 3.92it/s]
epoch 4 test acc 0.9369939271255061
100% 414/414 [05:07<00:00, 1.78it/s]
epoch 5 batch id 1 loss 0.002709627617150545 train acc 1.0
epoch 5 batch id 201 loss 0.015021243132650852 train acc 0.9877953980099502
epoch 5 batch id 401 loss 0.006084760185331106 train acc 0.9906094139650873
epoch 5 train acc 0.9904136473429952
100% 104/104 [00:27<00:00, 4.01it/s]
epoch 5 test acc 0.9380456098178137
```

test set 정확도

: 약 0.93

사용자를 위한 프로그램

문제상황

수많은 카페 리뷰를
모두 확인할 수 없다

기대 효과

가장 **긍정**적인 리뷰,
가장 **부정**적인 리뷰만 보면
가고싶은 카페의 장단점을
한눈에 파악할 수 있겠다!

사용자를 위한 프로그램

```
import selenium
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time
import warnings
warnings.filterwarnings("ignore")

from bs4 import BeautifulSoup
import re
import time
import math

#드라이버 불러오기
path = 'chromedriver'
excutable_path = 'chromedriver.exe'
source_url = "https://map.kakao.com/"
driver = webdriver.Chrome(executable_path=excutable_path)

#인터넷 접속
driver.get(source_url)
```

사용자를 위한 프로그램

```
import selenium
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time
import warnings
warnings.filterwarnings("ignore")

from bs4 import BeautifulSoup
import re
import time
import math

#드라이버 불러오
path = 'chromedr
excutable_path =
source_url = "ht
driver = webdriv

#인터넷 접속
driver.get(sourc
```

```
import torch

from transformers import BertTokenizer, BertForSequenceClassification, AdamW, BertConfig, get_linear_schedule_with_warmup
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler
from keras_preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split

import numpy as np
import random
import time
import datetime
```

사용자를 위한 프로그램

_입력받기

```
#사용자 입력 받기
place=input("리뷰를 보고 싶은 가게의 이름이 무엇인가요?")

#서치
search=driver.find_element(By.XPATH,'//*[@id="search.keyword.query"]')
search.send_keys(place)
search.send_keys(Keys.ENTER)

driver.implicitly_wait(time_to_wait=5)

#카페 이름, url 뜯어오기

urls=[]
names=[]


def name_url():

    html = driver.page_source
    soup = BeautifulSoup(html, "html.parser")
    moreviews = soup.find_all(name="a", attrs={"class":"moreview"}) #상세보기 모두 가져오기.
    namelist=soup.find_all(name='a',attrs={"class":"link_name"}) #식당 이름 가져오기
    #근데 스크롤 안 내린 거 상에서만 다 가져온 것.
    # < 눌러서 더 가져와야 함.

    for i in moreviews:
        page_url=i.get("href")
        urls.append(page_url)

    for i in namelist:
        name=i.get("title")
        names.append(name)

name_url() #첫 페이지에서 이름 뜯어옴.
```

리뷰를 보고 싶은 가게의 이름이 무엇인가요?블루보틀 성수

이후, 앞과 동일한 크롤링&전처리 과정을 거침

사용자를 위한 프로그램

모델 불러오기

```
model=BertForSequenceClassification.from_pretrained("C:/Users/sycy0/숙탯_딥러닝_플젝/bert_initial.h5")
```

모델에 넣기 위한 전처리

```
tokenizer=BertTokenizer.from_pretrained("bert-base-multilingual-cased",do_lower_case=False)
#다국어 버전 모델 내에 있는 토크나이저 사용/ 대소문자 구분
```

```
def convert_data(raw):
    review_bert=["[CLS] "+str(s)+" [SEP]" for s in raw.review]
    tokenized_texts=[tokenizer.tokenize(s) for s in review_bert]

    MAX_LEN = 128
    input_ids = [tokenizer.convert_tokens_to_ids(x) for x in tokenized_texts]
    input_ids = pad_sequences(input_ids, maxlen=MAX_LEN, dtype='long', truncating='post', padding='post')

    attention_masks=[]
    for seq in input_ids:
        seq_mask=[float(i>0) for i in seq] #seq가 0이 아니면, 어텐션 마스크에 1을 저장한다.
        attention_masks.append(seq_mask)

    final_input=torch.tensor(input_ids)
    final_mask=torch.tensor(attention_masks)

    return final_input, final_mask

def test_review(review):
    model.eval()
    inputs,masks=convert_data(review)
    b_input_ids=inputs
    b_input_mask=masks

    with torch.no_grad():
        outputs=model(b_input_ids,token_type_ids=None, attention_mask=b_input_mask)

    #예측하기
    logits=outputs.logits
    return logits
```

사용자를 위한 프로그램

_긍정 확률 구하기

```
pd.set_option('display.max_colwidth', -1)


logits=test_review(df_5)

import torch.nn.functional as F
probabilities = F.softmax(logits, dim=-1)

prob_list=((probabilities[:,0]*100).tolist()) #반올림, 뒤에 %붙이기 필요

df_5["positive_prob"]=prob_list

df_5.sort_values(by="positive_prob",ascending=False,inplace=True)
```

사용자를 위한 프로그램

_결과도출

```
print(place,"의 리뷰 중 긍정적인 리뷰는 다음과 같습니다.")
df_5.head(5)
```

블루보틀 성수 의 리뷰 중 긍정적인 리뷰는 다음과 같습니다.

	name	review	star	review_num	ave_star	positive_prob
139	블루보틀 성수 카페	주차 공간 너무 좋고 친절하고 쾌적하고 커피 맛도 좋았습니다 제 커피 내리면서 집중하고 직접 가져다 주시는데 이게 뭐라고 감동이죠	5.0	239	3.5	99.957832
63	블루보틀 성수 카페	핸드드립전문점 로스팅전문 드립커피 아메리카노 라떼 다 너무 좋음자리는 늘 부족	5.0	90	3.7	99.956444
5	블루보틀 성수 카페	놀라 아스크림 쫀쫀 고소 맛있어용	3.0	414	3.9	99.955444
153	블루보틀 성수 카페	직원분들 정말 친절하시고 핸드드립도 맛있음	5.0	16	3.8	99.955025
252	블루보틀 성수 카페	넘친절하게 설명도잘해주시고 세심함을 느꼈습니다	5.0	235	3.7	99.954910

사용자를 위한 프로그램

_결과도출

```
print(place,"의 리뷰 중 부정적인 리뷰는 다음과 같습니다.")
df_5.tail(5)
```

블루보틀 성수 의 리뷰 중 부정적인 리뷰는 다음과 같습니다.

	name	review	star	review_num	ave_star	positive_prob
414	블루보틀 성수 카페	스벅 같게	2.0	88	2.4	0.289019
94	블루보틀 성수 카페	집이 먼데 선물용이라 쇼핑백 구입한대도 안줌도쿄 블루보틀이 헐씬낫다육천원이 아까운 맛 맛없음	1.0	8	3.0	0.203658
193	블루보틀 성수 카페	드립이 먼저 제조됐는데 같이 주문한 아메리카노 순서가 아직 안됐다고 더 기다려 같이 받아야 된다 함 향 날아가고 식으며 드립 메리트 없어짐 아메리카노는 물양이 적다고 미리 양해 구하길래 뜨거운물 같이 달라니 안된다고 커피 다 마시고 나서 요청하면 줄 수 있다 함 위장에서 섞어야 함쟁반도 못 봄 컵만 들고 날라야 함그냥 모든 것이 회사입장에서 원칙이 적용되는 느낌직원들의 응대는 아주 친절하나	1.0	54	3.1	0.166801
231	블루보틀 성수 카페	커피맛없음 너무비쌈 자리불편 주차불가 커피빈 압승	1.0	14	3.3	0.119680
93	블루보틀 성수 카페	와이파이도 안되고 인테리어도 별로	2.0	119	3.7	0.117849

한계점

·BERT모델 공부

-BERT를 구성하는 다양한 기술들을 모두 공부해야 했다는 점
-BERT에 사용되는 모델과 라이브러리들이 모두 생소

·긍정/부정 라벨링

-긍/부정이 명확하지 않은 중립 리뷰를 어떻게 처리 할 것인가
-같은 평점이더라도 개개인에 따라 평가가 다를 수 있기에,
이를 어떻게 처리 할 것인가