Information Search Project

B813005 고민재

Subject

에브리타임 홍.맛.게 Sentiment Analysis

식당 리뷰 글을 문장 단위로 분석해서 긍정/부정으로 분류해보자

대학생만을 위한 커뮤니티

에브리타임

에브리타임 <야 넌 홍대생인데 홍대맛집도 모르냐-를 방지하기 위한 홍대맛집게시판> 크롤링

Keras RNN으로 네이버 영화 리뷰 감성 분석

Hugging Face의 PyTorch BERT로 모델 만들어 전이 학습

Dataset

Naver sentiment movie corpus v1.0

This is a movie review dataset in the Korean language. Reviews were scraped from Naver Movies.

The dataset construction is based on the method noted in Large movie review dataset from Maas et al., 2011.

Data description

- Each file is consisted of three columns: `id`, `document`, `label`
  - `id`: The review id, provieded by Naver
  - `document`: The actual review
  - `label`: The sentiment class of the review. (0: negative, 1: positive)
  - Columns are delimited with tabs (i.e., `.tsv` format; but the file extension is `.txt` for easy access for novices)
- 200K reviews in total
  - `ratings.txt`: All 200K reviews
  - `ratings_test.txt`: 50K reviews held out for testing
  - `ratings_train.txt`: 150K reviews for training

Characteristics

- All reviews are shorter than 140 characters
- Each sentiment class is sampled equally (i.e., random guess yields 50% accuracy)
  - 100K negative reviews (originally reviews of ratings 1-4)
  - 100K positive reviews (originally reviews of ratings 9-10)
  - Neutral reviews (originally reviews of ratings 5-8) are excluded

- BERT

Train 150000 Validation 10%
Test 50000

- RNN

Train 150000 Validation 10%
Test 50000

Dataset

```
]
df_raw = pd.read_csv("matge.csv", encoding="cp949")
```

```
] df_raw.info
```

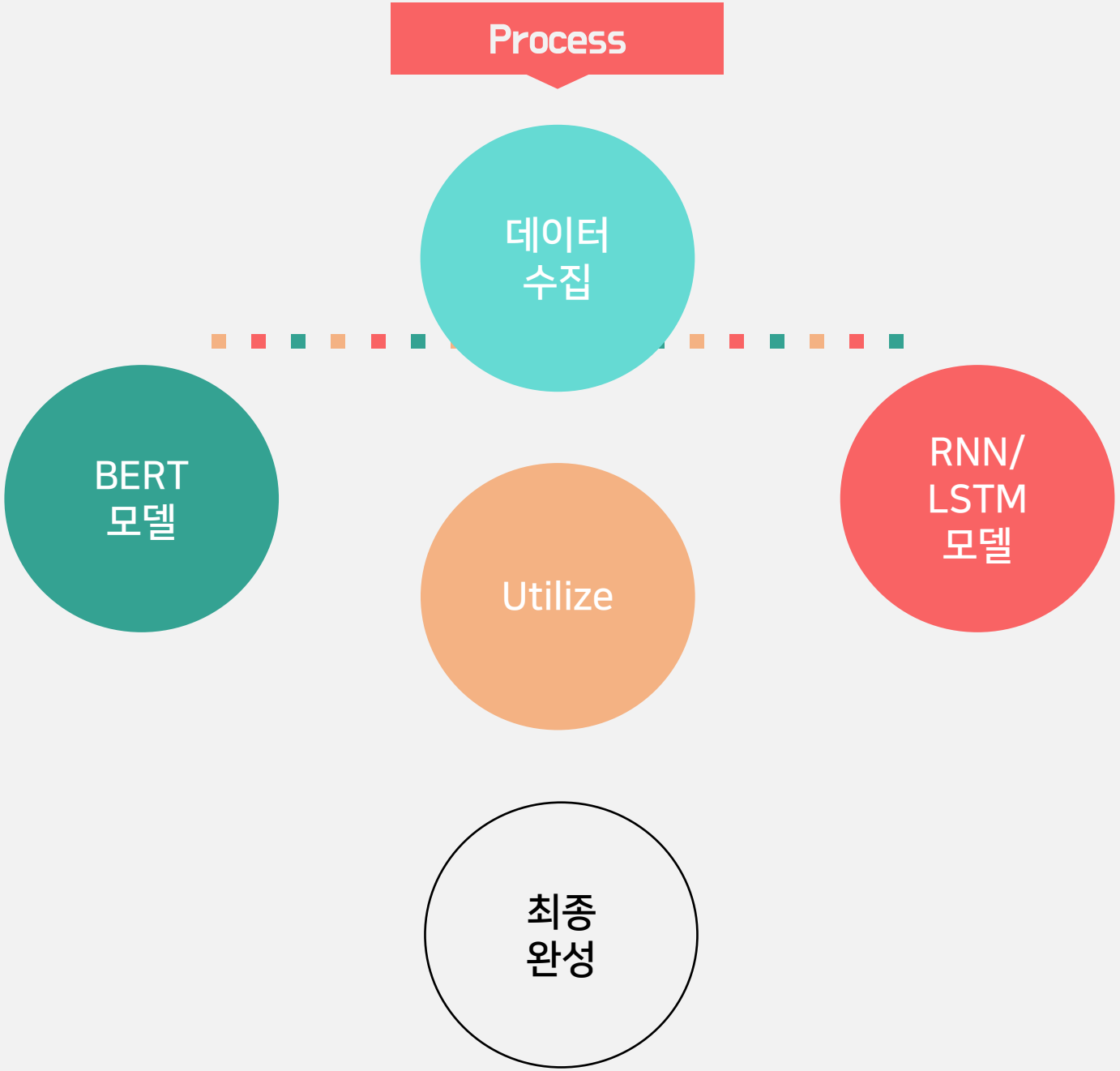
```
<bound method DataFrame.info of          식당명                                                리뷰 문장
0     레스토랑 아진                                연남동쪽에 있고 파스타 1.5 부채살 스테끼3.4
1     레스토랑 아진                          와인까지 세트로 5.4 저 감자 뭐시기는 0.5 레몬맛+소금맛
2     레스토랑 아진  알리오올리오인데 우리가 흔히 먹는 맛은 아니고 뭔가 치즈맛인가 맛표현 어려운데 맛있...
3     레스토랑 아진  스테끼는 뭐 당근퓨레랑 저 풀때기 뭔지 모르는데 같이 삭 해먹으면 아주 부드러움. ...
4     레스토랑 아진  남친이랑 100일 기념으로 갔는데 예약을 많이 하고 가는 듯. 예약 선입금 2만원 ...
...       ...                                                  ...
5304      NaN                                                  NaN
5305      NaN                                          재방문 의사 있음
5306      NaN                                                  NaN
5307      NaN                        (기본토스트보다는 돈 조금 보태서 햄이나 베이컨들어간걸 추천)
5308      NaN                                                  ]"["

[5309 rows x 2 columns]>
```

```
df_raw.sample(10)
```

	식당명	리뷰 문장
5023	당인동 국수공장	ㅡ넓음
1001	정문 앞 롤링파스타	다른 지점보다 가지라던지 재료가 조금 덜 들어있었지만
3692	홍아지트	NaN
1964	플라워	그리고 화장실이 깨끗하다!! <- 중요
2110	베리메리	NaN
3941	NaN	NaN
1319	NaN	]"["
4940	오키나와루	김부각에 와사비를 섞었는데 달아진 맛

- 게시글 2000개
- 문장 5309개
- Null제거 3298개

Platform


Process
데이터
수집
BERT
모델
Utilize
RNN/
LSTM
모델
최종
완성

Model

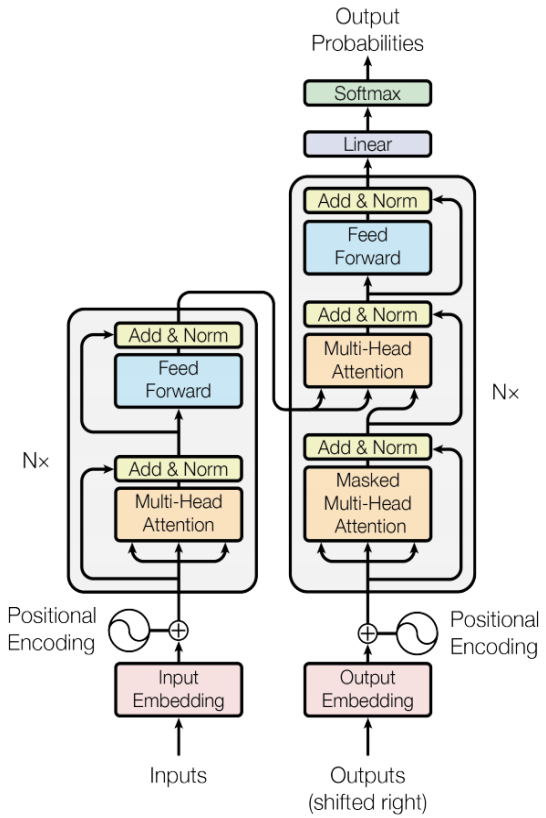
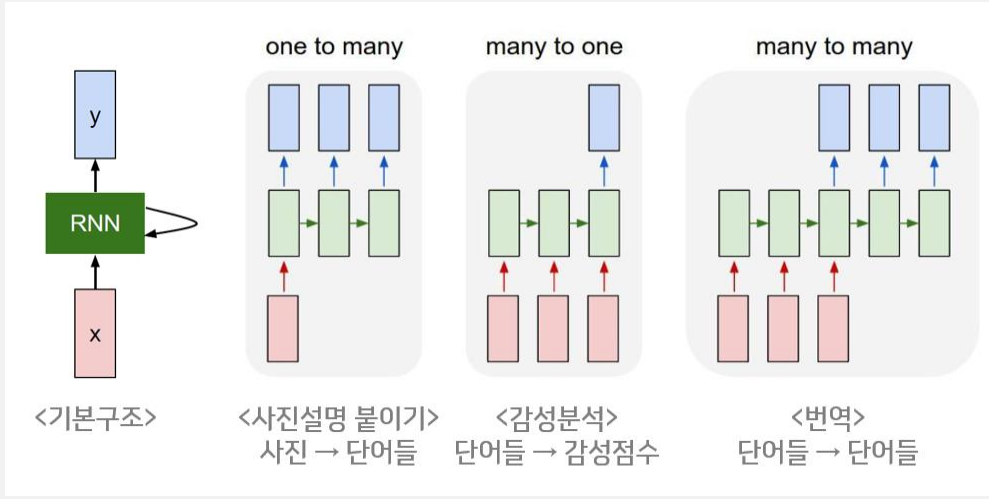
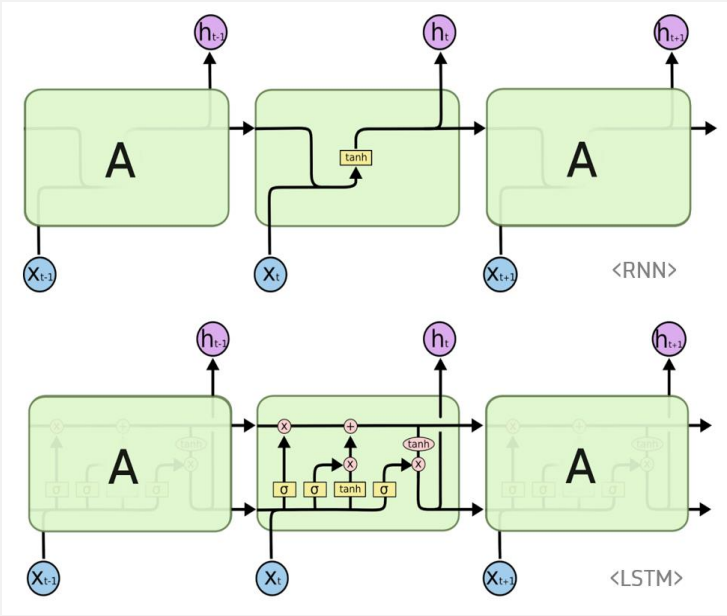


Figure 1: The Transformer - model architecture.





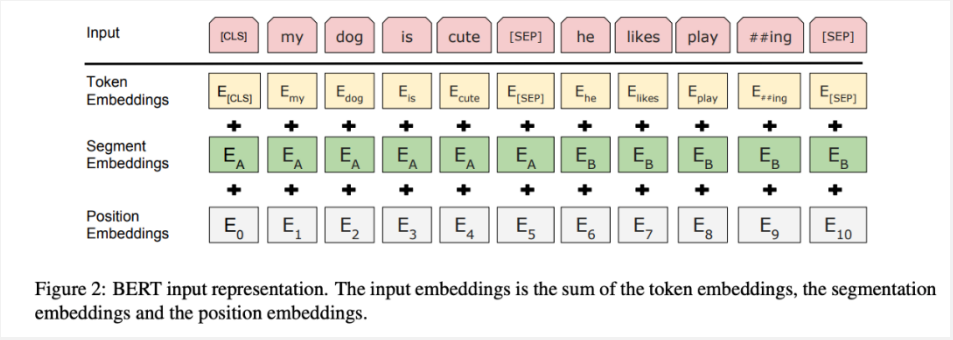
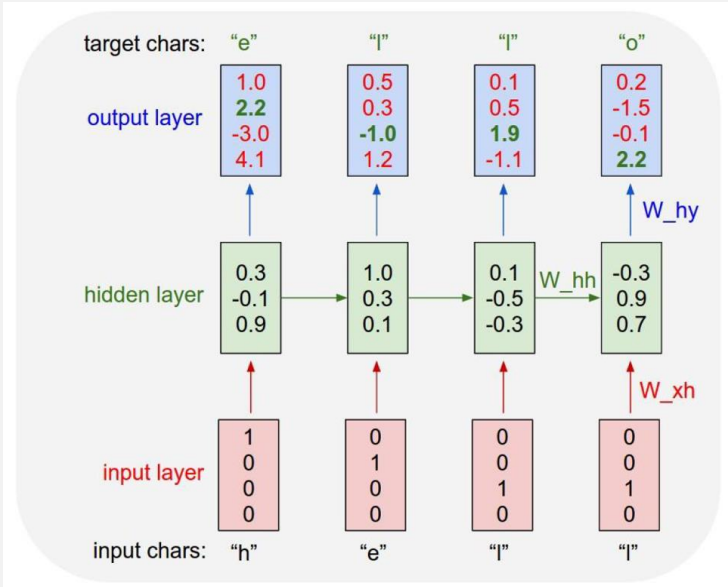
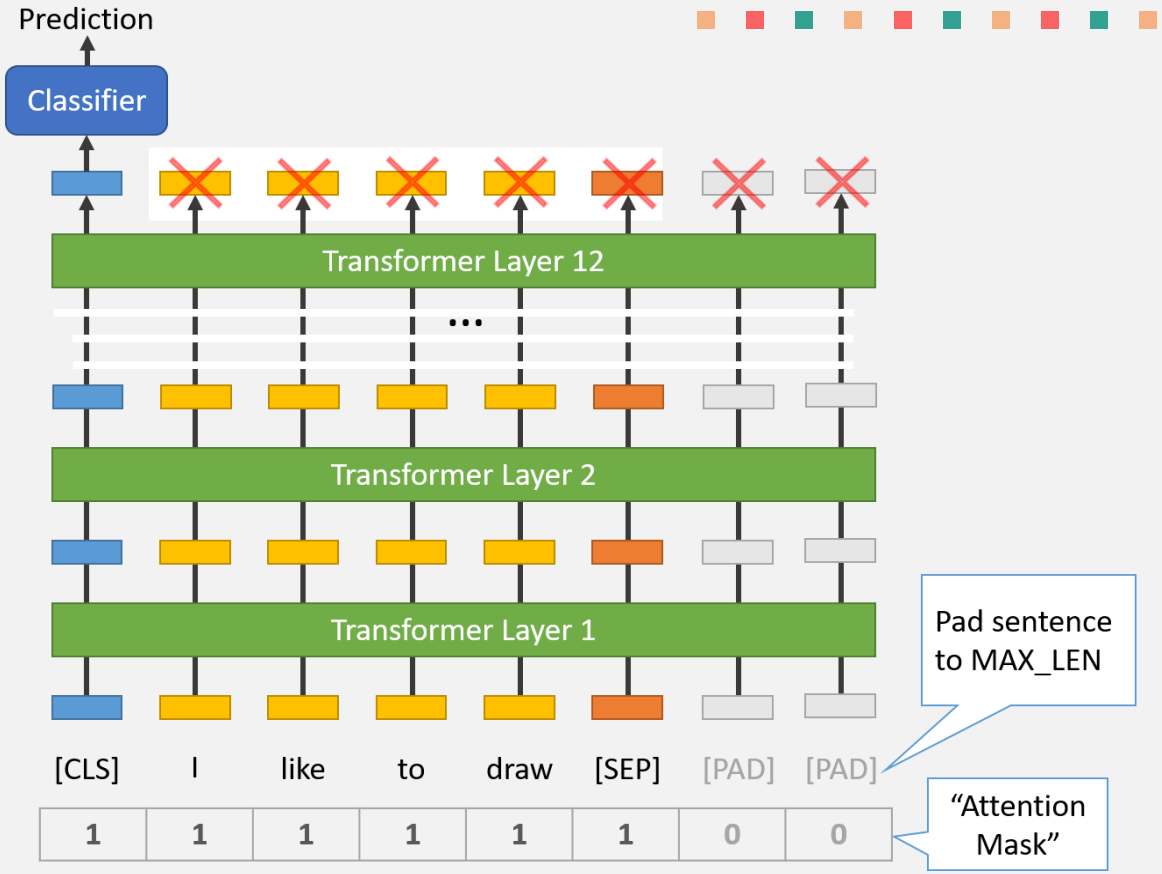


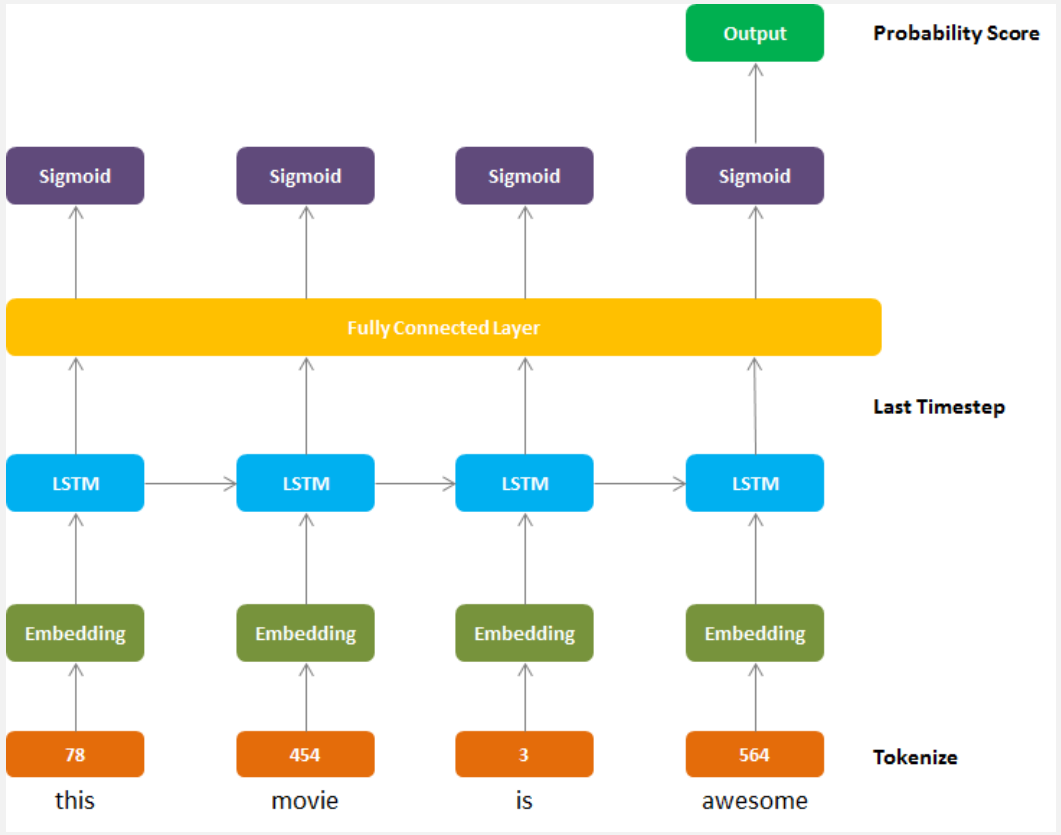
Figure 2: BERT input representation. The input embeddings is the sum of the token embeddings, the segmentation embeddings and the position embeddings.



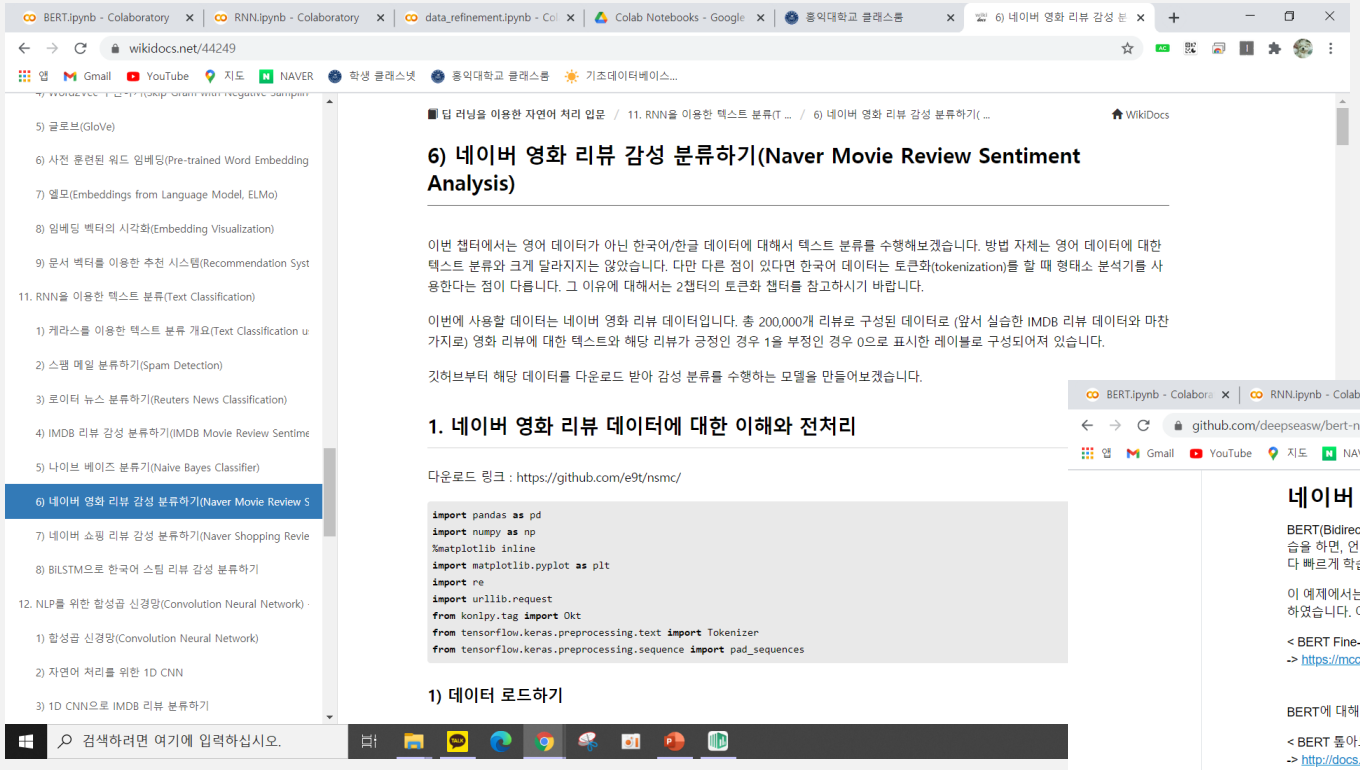
- 사전훈련된 BERT는 다양한 문제로 전이학습 가능
- 영화리뷰 문장이 입력으로 들어가면, 긍정/부정으로 구분
- 모델의 출력에서 [CLS] 위치인 첫 번째 토큰에 새로운 레이어를 붙여서 파인튜닝
- Huggning Face의 BertForSequenceClassification() 함수 이용 구현

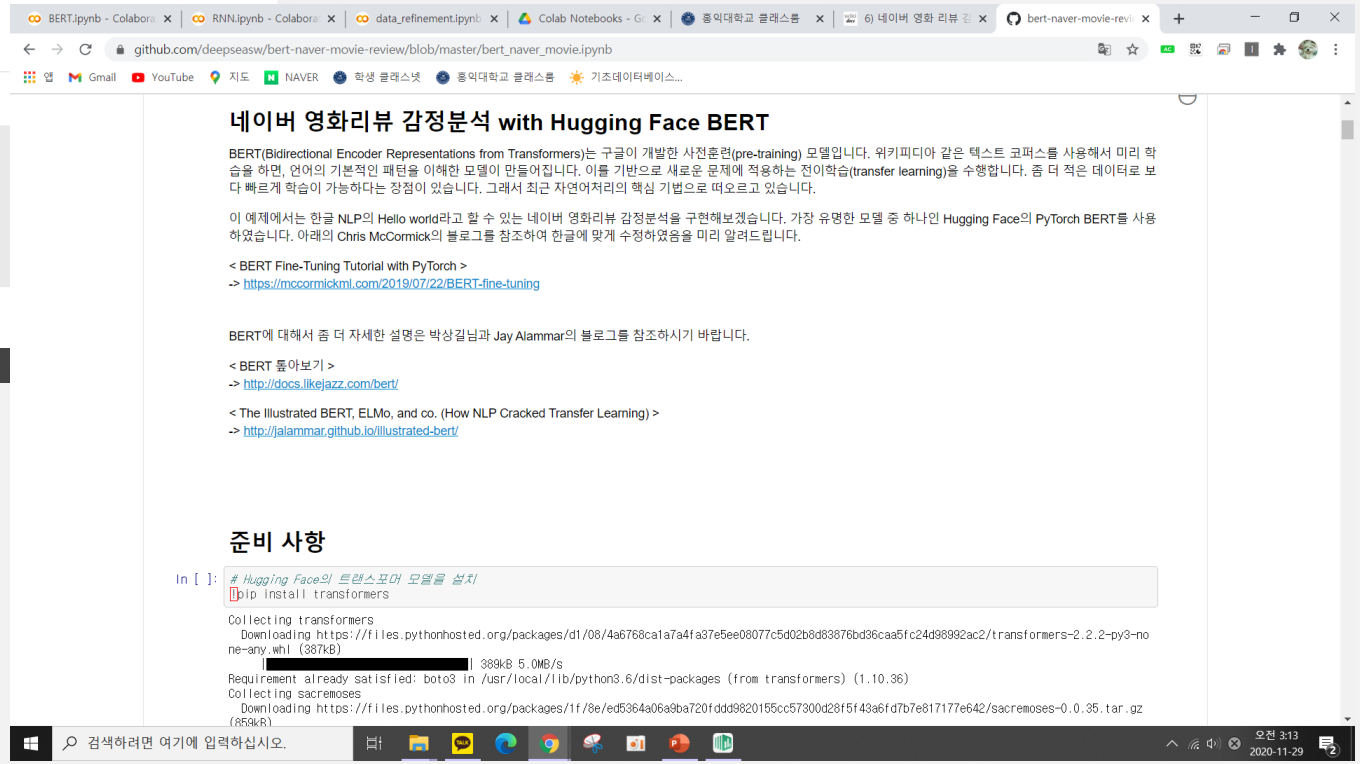
Training





Quotation


BERT.ipynb - Colaboratory
RNN.ipynb - Colaboratory
data_refinement.ipynb - Col
Colab Notebooks - Google
홍익대학교 클래스룸
6) 네이버 영화 리뷰 감성 분
wikidocs.net/44249
Gmail
YouTube
지도
NAVER
학생 클래스넷
홍익대학교 클래스룸
기초데이터베이스...
5) 글로브(GloVe)
6) 사전 훈련된 워드 임베딩(Pre-trained Word Embedding
7) 엘모(Embeddings from Language Model, ELMo)
8) 임베딩 벡터의 시각화(Embedding Visualization)
9) 문서 벡터를 이용한 추천 시스템(Recommendation Syst
11. RNN을 이용한 텍스트 분류(Text Classification)
1) 케라스를 이용한 텍스트 분류 개요(Text Classification u
2) 스팸 메일 분류하기(Spam Detection)
3) 로이터 뉴스 분류하기(Reuters News Classification)
4) IMDB 리뷰 감성 분류하기(IMDB Movie Review Sentime
5) 나이브 베이즈 분류기(Naive Bayes Classifier)
6) 네이버 영화 리뷰 감성 분류하기(Naver Movie Review S
7) 네이버 쇼핑 리뷰 감성 분류하기(Naver Shopping Revie
8) BiLSTM으로 한국어 스팀 리뷰 감성 분류하기
12. NLP를 위한 합성곱 신경망(Convolution Neural Network)
1) 합성곱 신경망(Convolution Neural Network)
2) 자연어 처리를 위한 1D CNN
3) 1D CNN으로 IMDB 리뷰 분류하기
딥 러닝을 이용한 자연어 처리 입문 / 11. RNN을 이용한 텍스트 분류(T ... / 6) 네이버 영화 리뷰 감성 분류하기(...
WikiDocs
6) 네이버 영화 리뷰 감성 분류하기(Naver Movie Review Sentiment Analysis)
이번 챕터에서는 영어 데이터가 아닌 한국어/한글 데이터에 대해서 텍스트 분류를 수행해보겠습니다. 방법 자체는 영어 데이터에 대한 텍스트 분류와 크게 달라지지는 않았습니다. 다만 다른 점이 있다면 한국어 데이터는 토큰화(tokenization)를 할 때 형태소 분석기를 사용한다는 점이 다릅니다. 그 이유에 대해서는 2챕터의 토큰화 챕터를 참고하시기 바랍니다.
이번에 사용할 데이터는 네이버 영화 리뷰 데이터입니다. 총 200,000개 리뷰로 구성된 데이터로 (앞서 실습한 IMDB 리뷰 데이터와 마찬가지로) 영화 리뷰에 대한 텍스트와 해당 리뷰가 긍정인 경우 1을 부정인 경우 0으로 표시한 레이블로 구성되어져 있습니다.
깃허브부터 해당 데이터를 다운로드 받아 감성 분류를 수행하는 모델을 만들어보겠습니다.
1. 네이버 영화 리뷰 데이터에 대한 이해와 전처리
다운로드 링크 : https://github.com/e9t/nsmc/
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import re
import urllib.request
from konlpy.tag import Okt
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
1) 데이터 로드하기
검색하려면 여기에 입력하십시오.


BERT.ipynb - Colabora
RNN.ipynb - Colabora
data_refinement.ipynb
Colab Notebooks - Go
홍익대학교 클래스룸
6) 네이버 영화 리뷰 감
bert-naver-movie-revi
github.com/deepseasw/bert-naver-movie-review/blob/master/bert_naver_movie.ipynb
Gmail
YouTube
지도
NAVER
학생 클래스넷
홍익대학교 클래스룸
기초데이터베이스...
네이버 영화리뷰 감정분석 with Hugging Face BERT
BERT(Bidirectional Encoder Representations from Transformers)는 구글이 개발한 사전훈련(pre-training) 모델입니다. 위키피디아 같은 텍스트 코퍼스를 사용해서 미리 학습을 하면, 언어의 기본적인 패턴을 이해한 모델이 만들어집니다. 이를 기반으로 새로운 문제에 적용하는 전이학습(transfer learning)을 수행합니다. 좀 더 적은 데이터로 보다 빠르게 학습이 가능하다는 장점이 있습니다. 그래서 최근 자연어처리의 핵심 기법으로 떠오르고 있습니다.
이 예제에서는 한글 NLP의 Hello world라고 할 수 있는 네이버 영화리뷰 감정분석을 구현해보겠습니다. 가장 유명한 모델 중 하나인 Hugging Face의 PyTorch BERT를 사용하였습니다. 아래의 Chris McCormick의 블로그를 참조하여 한글에 맞게 수정하였음을 미리 알려드립니다.
< BERT Fine-Tuning Tutorial with PyTorch >
-> https://mccormickml.com/2019/07/22/BERT-fine-tuning
BERT에 대해서 좀 더 자세한 설명은 박상길님과 Jay Alammar의 블로그를 참조하시기 바랍니다.
< BERT 톺아보기 >
-> http://docs.likejazz.com/bert/
< The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning) >
-> http://jalammar.github.io/illustrated-bert/
준비 사항
In [ ]:
Hugging Face의 트랜스포머 모델을 설치
!pip install transformers
Collecting transformers
Downloading https://files.pythonhosted.org/packages/d1/08/4a6768ca1a7a4fa37e5ee08077c5d02b8d83876bd36caa5fc24d98992ac2/transformers-2.2.2-py3-none-any.whl (387kB)
389kB 5.0MB/s
Requirement already satisfied: boto3 in /usr/local/lib/python3.6/dist-packages (from transformers) (1.10.36)
Collecting sacremoses
Downloading https://files.pythonhosted.org/packages/1f/8e/ed5364a06a9ba720fddd9820155cc57300d28f5f43a6fd7b7e817177e642/sacremoses-0.0.35.tar.gz
검색하려면 여기에 입력하십시오.
오전 3:13
2020-11-29

Contribution

- 기존의 정제된 Dataset을 가져다 결과를 도출한 것이 아니라 직접 수집하고 분류하는 과정을 거쳤음
- 실생활과 맞닿은 프로젝트를 하고자 했던 목적을 달성

Contribution

```
======== Epoch 1 / 3 ========
Training...
  Batch   500  of  4,219.    Elapsed: 0:05:46.
  Batch 1,000  of  4,219.    Elapsed: 0:11:33.
  Batch 1,500  of  4,219.    Elapsed: 0:17:19.
  Batch 2,000  of  4,219.    Elapsed: 0:23:06.
  Batch 2,500  of  4,219.    Elapsed: 0:28:53.
  Batch 3,000  of  4,219.    Elapsed: 0:34:40.
  Batch 3,500  of  4,219.    Elapsed: 0:40:26.
  Batch 4,000  of  4,219.    Elapsed: 0:46:13.

  Average training loss: 0.17
  Training epcoh took: 0:48:44

Running Validation...
  Accuracy: 13.07
  Validation took: 0:01:50
```

```python
# 정확도 계산 함수
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)
```

Validation Accuracy 값에 이상
-> 한 epoch 돌아야 확인이 가능하여 해당 오류를 확인하고 수정하는데 많은 시간 소요 있었음
정의한 정확도 계산 함수에 오류가 있었음을 확인하고 수정

```
======== Epoch 3 / 3 ========
Training...
  Batch   500  of  4,219.    Elapsed: 0:05:47.
  Batch 1,000  of  4,219.    Elapsed: 0:11:34.
  Batch 1,500  of  4,219.    Elapsed: 0:17:21.
  Batch 2,000  of  4,219.    Elapsed: 0:23:08.
  Batch 2,500  of  4,219.    Elapsed: 0:28:55.
  Batch 3,000  of  4,219.    Elapsed: 0:34:41.
  Batch 3,500  of  4,219.    Elapsed: 0:40:28.
  Batch 4,000  of  4,219.    Elapsed: 0:46:14.

  Average training loss: 0.19
  Training epcoh took: 0:48:46

Running Validation...
  Accuracy: 0.87
  Validation took: 0:01:49

Training complete!
```

기존 source에서 epoch=4
-> 시간 단축을 위해 epoch=3으로 진행해보았으나
Loss와 accuracy에서 뒤처지지 않는 것을 확인

Contribution

```
#0은 부정 1은 긍정
def good_or_bad(sentence):
  logits = test_sentences([sentence])
  determine = np.argmax(logits)

  return determine
```

```
#극찬 영화평 BERT로 확인해보기
good_or_bad("와 개쩐다 정말 세계관 최강자들의 영화다")
```

```
1
```

```
#리뷰 라벨링
def label_review_bert(series_object):
  list_scores=[]
  list_reviews = series_object.to_list()
  for item in list_reviews:
    score = good_or_bad(item)
    list_scores.append(score)
  return list_scores
```

```
df_dropped["BERT_SCORE"] = label_review_bert(df_dropped["리뷰 문장"])
```

Softmax가 적용되지않은 출력 logits에 대해 argmax로 더 높은 값을 라벨로 설정하여 긍정, 부정을 분류하는 함수를 만들어 모든 리뷰에 적용하여 labeling

```
def label_review(series_object):
    list_scores = []
    list_reviews = series_object.to_list()
    for item in list_reviews:
        item = okt.morphs(item, stem=True) # 토큰화
        item = [word for word in item if not word in stopwords] # 불용어 제거
        encoded = tokenizer.texts_to_sequences([item]) # 정수 인코딩
        pad_new = pad_sequences(encoded, maxlen = max_len) # 패딩
        score = float(loaded_model.predict(pad_new)) # 예측
        list_scores.append(score)
    return list_scores
```

```
df_rnn["RNN_SCORE"] = label_review(df_rnn["리뷰 문장"])
df_rnn.sample(10)
```

Result

BERT

```
======== Epoch 1 / 3 ========
Training...
  Batch   500  of  4,219.    Elapsed: 0:05:46.
  Batch 1,000  of  4,219.    Elapsed: 0:11:32.
  Batch 1,500  of  4,219.    Elapsed: 0:17:18.
  Batch 2,000  of  4,219.    Elapsed: 0:23:03.
  Batch 2,500  of  4,219.    Elapsed: 0:28:48.
  Batch 3,000  of  4,219.    Elapsed: 0:34:34.
  Batch 3,500  of  4,219.    Elapsed: 0:40:19.
  Batch 4,000  of  4,219.    Elapsed: 0:46:04.

  Average training loss: 0.08
  Training epcoh took: 0:48:35

Running Validation...
  Accuracy: 0.86
  Validation took: 0:01:49
```

```
======== Epoch 2 / 3 ========
Training...
  Batch   500  of  4,219.    Elapsed: 0:05:46.
  Batch 1,000  of  4,219.    Elapsed: 0:11:32.
  Batch 1,500  of  4,219.    Elapsed: 0:17:18.
  Batch 2,000  of  4,219.    Elapsed: 0:23:04.
  Batch 2,500  of  4,219.    Elapsed: 0:28:50.
  Batch 3,000  of  4,219.    Elapsed: 0:34:37.
  Batch 3,500  of  4,219.    Elapsed: 0:40:23.
  Batch 4,000  of  4,219.    Elapsed: 0:46:09.

  Average training loss: 0.11
  Training epcoh took: 0:48:40

Running Validation...
  Accuracy: 0.87
  Validation took: 0:01:50
```

```
======== Epoch 3 / 3 ========
Training...
  Batch   500  of  4,219.    Elapsed: 0:05:47.
  Batch 1,000  of  4,219.    Elapsed: 0:11:34.
  Batch 1,500  of  4,219.    Elapsed: 0:17:21.
  Batch 2,000  of  4,219.    Elapsed: 0:23:08.
  Batch 2,500  of  4,219.    Elapsed: 0:28:55.
  Batch 3,000  of  4,219.    Elapsed: 0:34:41.
  Batch 3,500  of  4,219.    Elapsed: 0:40:28.
  Batch 4,000  of  4,219.    Elapsed: 0:46:14.

  Average training loss: 0.19
  Training epcoh took: 0:48:46

Running Validation...
  Accuracy: 0.87
  Validation took: 0:01:49

Training complete!
```

```
Batch  100 of 1,563. Elapsed: 0:00:00.
Batch  200 of 1,563. Elapsed: 0:00:00.
Batch  300 of 1,563. Elapsed: 0:00:01.
Batch  400 of 1,563. Elapsed: 0:00:01.
Batch  500 of 1,563. Elapsed: 0:00:01.
Batch  600 of 1,563. Elapsed: 0:00:01.
Batch  700 of 1,563. Elapsed: 0:00:02.
Batch  800 of 1,563. Elapsed: 0:00:02.
Batch  900 of 1,563. Elapsed: 0:00:02.
Batch1,000 of 1,563. Elapsed: 0:00:02.
Batch1,100 of 1,563. Elapsed: 0:00:03.
Batch1,200 of 1,563. Elapsed: 0:00:03.
Batch1,300 of 1,563. Elapsed: 0:00:03.
Batch1,400 of 1,563. Elapsed: 0:00:03.
Batch1,500 of 1,563. Elapsed: 0:00:04.

Accuracy: 0.89
Validation took: 0:00:04
```

Result

LSTM

```
1532/1532 [==============================] - 26s 17ms/step - loss: 0.3350 - acc: 0.8563

테스트 정확도: 0.8563
```

```
Epoch 1/15
2181/2181 [==============================] - ETA: 0s - loss: 0.3862 - acc: 0.8245
Epoch 00001: val_acc improved from -inf to 0.85094, saving model to best_model.h5
2181/2181 [==============================] - 257s 118ms/step - loss: 0.3862 - acc: 0.8245 - val_loss: 0.3417 - val_acc: 0.8509
Epoch 2/15
2181/2181 [==============================] - ETA: 0s - loss: 0.3238 - acc: 0.8598
Epoch 00002: val_acc improved from 0.85094 to 0.85479, saving model to best_model.h5
2181/2181 [==============================] - 256s 118ms/step - loss: 0.3238 - acc: 0.8598 - val_loss: 0.3301 - val_acc: 0.8548
Epoch 3/15
2181/2181 [==============================] - ETA: 0s - loss: 0.2995 - acc: 0.8735
Epoch 00003: val_acc improved from 0.85479 to 0.86188, saving model to best_model.h5
2181/2181 [==============================] - 256s 117ms/step - loss: 0.2995 - acc: 0.8735 - val_loss: 0.3228 - val_acc: 0.8619
Epoch 4/15
2181/2181 [==============================] - ETA: 0s - loss: 0.2821 - acc: 0.8826
Epoch 00004: val_acc improved from 0.86188 to 0.86298, saving model to best_model.h5
2181/2181 [==============================] - 257s 118ms/step - loss: 0.2821 - acc: 0.8826 - val_loss: 0.3172 - val_acc: 0.8630
Epoch 5/15
2181/2181 [==============================] - ETA: 0s - loss: 0.2666 - acc: 0.8902
Epoch 00005: val_acc improved from 0.86298 to 0.86346, saving model to best_model.h5
2181/2181 [==============================] - 255s 117ms/step - loss: 0.2666 - acc: 0.8902 - val_loss: 0.3186 - val_acc: 0.8635
Epoch 6/15
2181/2181 [==============================] - ETA: 0s - loss: 0.2518 - acc: 0.8970
Epoch 00006: val_acc improved from 0.86346 to 0.86353, saving model to best_model.h5
2181/2181 [==============================] - 254s 116ms/step - loss: 0.2518 - acc: 0.8970 - val_loss: 0.3212 - val_acc: 0.8635
Epoch 7/15
2181/2181 [==============================] - ETA: 0s - loss: 0.2363 - acc: 0.9049
Epoch 00007: val_acc did not improve from 0.86353
2181/2181 [==============================] - 255s 117ms/step - loss: 0.2363 - acc: 0.9049 - val_loss: 0.3335 - val_acc: 0.8600
Epoch 8/15
2181/2181 [==============================] - ETA: 0s - loss: 0.2212 - acc: 0.9120
Epoch 00008: val_acc did not improve from 0.86353
2181/2181 [==============================] - 253s 116ms/step - loss: 0.2212 - acc: 0.9120 - val_loss: 0.3352 - val_acc: 0.8569
Epoch 00008: early stopping
```

Result

```
df_res = pd.read_csv("HMG_Sentiment_Analysis_BERT_RNN.csv", encoding="cp949")
```

```
df_res.sample(10)
```

	식당명	리뷰 문장	BERT_SCORE	RNN_SCORE
1370	연남동 감칠	평소 비슷한 양식 메뉴에 질리면 한번은 먹으러 가도 좋을 듯 해요	0	0.964277
899	최강금 돈가스 (합정)	오픈형 주방이라 이불덮밥처럼 주방을 테이블이 둘러싸는형식 위생 걱정 없이 먹을 수 ...	1	0.948059
385	나물먹는곰	사장님 고객대응이 너무 최악이네	0	0.000961
760	서교순대국	참고로 내가 시킨건 순대국특임	0	0.093732
1610	코너스테이크	홍대생할인은 계산할때 얘기하면 해주고 7500원9500원 사이에서 다 먹을 수 있어...	1	0.990016
276	동대문 신 야채곱창 마포점	맛은 무난무난 근데 또시킬거같진않음	0	0.248073
3038	홍타이루 마라탕	위에 뿌려진 소스 잘 섞어서 먹으면 돼	0	0.278584
2244	홍대 코너스테이크	면이랑 소스랑 결별한게 분명함 서로 어울리질 못함	0	0.287775
350	별버거	반숙계란 추가 필수 노른자 터졌을때 빵 고기 야채랑 같이먹는 이 맛이란	1	0.835488
641	혜윰인핑크	웬만한 메뉴는 다 만들어주십니다 아메에 민트시럽 넣어달라고 했을때도 굉장히 의아해하...	0	0.216990

문제점

- 신조어, 축약어 분석에 어려움
  ex) 믿먹 유메, 헌하오츠
- 기존의 영화 리뷰가 감정에 기반하여 작성되는 경향이 큰 반면, 홍맛게 리뷰글에서는 정보 전달을 목적으로 작성된 문장이 많았음
  ex) 편육12000 들기름메밀면 9000
- 휴대폰으로 작성된 글이 대부분이라 게시글 내에서 문장 단위로 끊어져 있지 않은 경우가 많았음

활용자료

https://github.com/e9t/nsmc
https://github.com/deepseasw/bert-naver-movie-review
https://github.com/snoop2head/yonsei-exchange-program
https://wikidocs.net/44249
https://blog.yourssu.com/post/everytime-bot-1/
https://yeoulcoding.tistory.com/121
https://itsallgoodman.tistory.com/9
https://github.com/konlpy/konlpy/issues/288
http://docs.likejazz.com/bert/
https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

Thank You