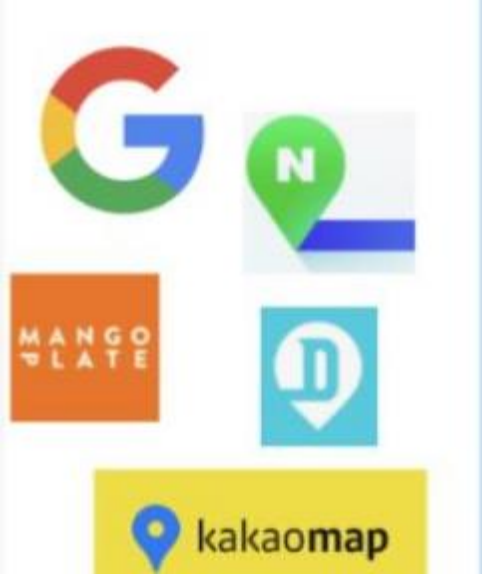
리뷰 감성분석 기반
홍대 맛집 추천 서비스

B731079 나민수
B735063 김범석
B811017 김성겸

목차

1. 주제 소개



- 리뷰 댓글에 대한 감성 분석을 통한 맛집 추천 서비스

기대효과 : 기존의 별점을 통한 모호한 기준의 맛집 추천보다는 리뷰 및 댓글 분석을 통해 신뢰성 있는 기준을 세워 더 나은 추천을 제공

2. 중간 발표 시 피드백 & 목표

보다 다양한 모델을
활용하여 돌려 볼 것

기존에 LSTM 모델을 통한 분석에서 Bidirectional LSTM, KNN, Logistic Regression 등 다양한 모델을 활용해 성능을 비교 & 분석

데이터의 양을 더욱 늘려서
시도해 볼 것
(기존 데이터: 리뷰 500개)

기존에 리뷰 500개를 활용해 분석하였지만 데이터의 양이 부족함을 느낌.
크롤링을 통해 데이터의 양 늘림.

3. 사용 모델 소개 및 모델별 성능 비교

데이터 양 : 500개 -> 1200개

3.1 Classical Machine Learning (KNN)

```
from sklearn.model_selection import KFold


params = {
  "n_neighbors" : list(range(1, 11))
}

# 사용할 모델 객체 생성
model1 = KNeighborsClassifier()

# 최적의 하이퍼 파라미터를 찾는다.
kfold = KFold(n_splits=10, shuffle=True, random_state = 1)
grid_clf1 = GridSearchCV(model1, param_grid = params, scoring="f1", cv=kfold)
grid_clf1.fit(x_train, y_train)

# 결과 출력
print(f"최적의 하이퍼 파라미터 : {grid_clf1.best_params_}")
print(f"최적의 모델 평균 성능 : {grid_clf1.best_score_}")
```

```
최적의 하이퍼 파라미터 : {'n_neighbors': 10}
최적의 모델 평균 성능 : 0.8396960124384769
```

그리드 서치를 통해
하이퍼파라미터 최적화

```
# knn 모델 평가
model = grid_clf1.best_estimator_
model.score(x_test, y_test)
```

```
0.7447916666666666
```

KNN 정확도 : 74.47%

3. 사용 모델 소개 및 모델별 성능 비교

데이터 양 : 500개 -> 1200개

3.2 Classical Machine Learning (Logistic Regression)

```
params = {
    'penalty' : ['l1', 'l2', 'elasticnet', 'none'],
    'C' : [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
}

model2 = LogisticRegression()

grid_clf2 = GridSearchCV(model2, param_grid=params, scoring='f1', cv=kfold)
grid_clf2.fit(x_train, y_train)

print(f'최적의 하이퍼 파라미터 : {grid_clf2.best_params_}')
print(f'최적의 모델 평균 성능 : {grid_clf2.best_score_}')
```

```
최적의 하이퍼 파라미터 : {'C': 10, 'penalty': 'l2'}
최적의 모델 평균 성능 : 0.852618522869118
```

그리드 서치를 통해
하이퍼파라미터 최적화

```
0] # logistic 모델 평가
   model = grid_clf2.best_estimator_
   model.score(x_test, y_test)
```

```
0.796875
```

Logistic Regression
정확도 : 79.68%

3. 사용 모델 소개 및 모델별 성능 비교

데이터 양 : 500개 -> 1200개

3.3 LSTM(Long Term Short Memory)

```
[58] history = model.fit(x_train, y_train, epochs=10, batch_size=10, validation_split=0.1)

Epoch 1/10
61/61 [==============================] - 3s 24ms/step - loss: 0.6316 - accuracy: 0.6595 - val_loss: 0.5637 - val_accuracy: 0.7463
Epoch 2/10
61/61 [==============================] - 1s 14ms/step - loss: 0.4878 - accuracy: 0.7940 - val_loss: 0.4983 - val_accuracy: 0.8060
Epoch 3/10
61/61 [==============================] - 1s 14ms/step - loss: 0.2604 - accuracy: 0.9020 - val_loss: 0.5087 - val_accuracy: 0.7910
Epoch 4/10
61/61 [==============================] - 1s 10ms/step - loss: 0.1448 - accuracy: 0.9468 - val_loss: 0.5900 - val_accuracy: 0.8060
Epoch 5/10
61/61 [==============================] - 1s 10ms/step - loss: 0.0956 - accuracy: 0.9718 - val_loss: 0.6678 - val_accuracy: 0.8060
Epoch 6/10
61/61 [==============================] - 1s 10ms/step - loss: 0.0623 - accuracy: 0.9817 - val_loss: 0.6948 - val_accuracy: 0.8209
Epoch 7/10
61/61 [==============================] - 1s 10ms/step - loss: 0.0480 - accuracy: 0.9917 - val_loss: 0.8181 - val_accuracy: 0.7761
Epoch 8/10
61/61 [==============================] - 1s 10ms/step - loss: 0.0452 - accuracy: 0.9900 - val_loss: 0.7879 - val_accuracy: 0.8209
Epoch 9/10
61/61 [==============================] - 1s 10ms/step - loss: 0.0397 - accuracy: 0.9867 - val_loss: 0.9391 - val_accuracy: 0.7910
Epoch 10/10
61/61 [==============================] - 1s 10ms/step - loss: 0.0297 - accuracy: 0.9934 - val_loss: 1.0419 - val_accuracy: 0.7910
```

```
[59] model.evaluate(x_test, y_test)

9/9 [==============================] - 0s 5ms/step - loss: 1.0621 - accuracy: 0.7700
[1.062145709991455, 0.7700348496437073]
```

LSTM
정확도 : 77.00%

3. 사용 모델 소개 및 모델별 성능 비교

데이터 양 : 500개 -> 1200개

3.4 Bidirectional LSTM

```
history = model.fit(x_train, y_train, epochs=10, batch_size=10, validation_split=0.1)

Epoch 1/10
61/61 [==============================] - 7s 41ms/step - loss: 0.6281 - acc: 0.6694 - val_loss: 0.5345 - val_acc: 0.7164
Epoch 2/10
61/61 [==============================] - 1s 21ms/step - loss: 0.4906 - acc: 0.8007 - val_loss: 0.4868 - val_acc: 0.8060
Epoch 3/10
61/61 [==============================] - 1s 16ms/step - loss: 0.3945 - acc: 0.8704 - val_loss: 0.5136 - val_acc: 0.7910
Epoch 4/10
61/61 [==============================] - 1s 16ms/step - loss: 0.2584 - acc: 0.9120 - val_loss: 0.6229 - val_acc: 0.7761
Epoch 5/10
61/61 [==============================] - 1s 17ms/step - loss: 0.1918 - acc: 0.9269 - val_loss: 0.6097 - val_acc: 0.8060
Epoch 6/10
61/61 [==============================] - 1s 16ms/step - loss: 0.1818 - acc: 0.9352 - val_loss: 0.7698 - val_acc: 0.7910
Epoch 7/10
61/61 [==============================] - 1s 16ms/step - loss: 0.1354 - acc: 0.9551 - val_loss: 0.7576 - val_acc: 0.8060
Epoch 8/10
61/61 [==============================] - 1s 16ms/step - loss: 0.1462 - acc: 0.9535 - val_loss: 0.7630 - val_acc: 0.8358
Epoch 9/10
61/61 [==============================] - 1s 16ms/step - loss: 0.0962 - acc: 0.9618 - val_loss: 0.7767 - val_acc: 0.8060
Epoch 10/10
61/61 [==============================] - 1s 16ms/step - loss: 0.0729 - acc: 0.9734 - val_loss: 0.8755 - val_acc: 0.7761
```

```
model.evaluate(x_test, y_test)

9/9 [==============================] - 0s 12ms/step - loss: 0.7830 - acc: 0.7526
[0.7829858064651489, 0.7526132464408875]
```

Bidirectional LSTM
정확도 : 75.26%

3. 사용 모델 소개 및 모델별 성능 비교

데이터 양 : 500개 -> 1200개

3.5 GRU (Gated Recurrent Unit)

```
history = model.fit(x_train, y_train, epochs=10, batch_size=10, validation_split=0.1)

Epoch 1/10
61/61 [==============================] - 3s 19ms/step - loss: 0.6109 - acc: 0.6578 - val_loss: 0.5346 - val_acc: 0.7313
Epoch 2/10
61/61 [==============================] - 1s 11ms/step - loss: 0.4590 - acc: 0.8007 - val_loss: 0.5364 - val_acc: 0.7612
Epoch 3/10
61/61 [==============================] - 1s 10ms/step - loss: 0.2887 - acc: 0.8854 - val_loss: 0.5795 - val_acc: 0.7910
Epoch 4/10
61/61 [==============================] - 1s 11ms/step - loss: 0.2052 - acc: 0.9136 - val_loss: 0.7420 - val_acc: 0.8209
Epoch 5/10
61/61 [==============================] - 1s 10ms/step - loss: 0.3674 - acc: 0.9319 - val_loss: 0.8437 - val_acc: 0.8060
Epoch 6/10
61/61 [==============================] - 1s 11ms/step - loss: 0.1491 - acc: 0.9369 - val_loss: 0.7578 - val_acc: 0.7910
Epoch 7/10
61/61 [==============================] - 1s 11ms/step - loss: 0.1142 - acc: 0.9535 - val_loss: 0.8380 - val_acc: 0.7313
Epoch 8/10
61/61 [==============================] - 1s 11ms/step - loss: 0.0852 - acc: 0.9668 - val_loss: 0.9477 - val_acc: 0.7463
Epoch 9/10
61/61 [==============================] - 1s 11ms/step - loss: 0.0737 - acc: 0.9718 - val_loss: 1.1440 - val_acc: 0.7761
Epoch 10/10
61/61 [==============================] - 1s 10ms/step - loss: 0.0622 - acc: 0.9817 - val_loss: 1.2090 - val_acc: 0.7612
```

```
[55] model.evaluate(x_test, y_test)

9/9 [==============================] - 0s 6ms/step - loss: 0.9076 - acc: 0.7909
[0.9076251983642578, 0.7909407615661621]
```

Gated Recurrent Unit
정확도 : 79.09%

3. 사용 모델 소개 및 모델별 성능 비교

데이터 양 : 500개 -> 1200개

3.6 KoBERT (Korean Bidirectional Encoder Representations from Transformers)

```
  4%|▍         | 1/24 [00:00<00:17,  1.35it/s]epoch 3 batch id 1 loss 0.6108501553535461 train acc 0.65625
100%|██████████| 24/24 [00:18<00:00,  1.33it/s]epoch 3 train acc 0.7165178571428572

100%|██████████| 6/6 [00:01<00:00,  3.65it/s]epoch 3 test acc 0.7708333333333334

  4%|▍         | 1/24 [00:00<00:17,  1.32it/s]epoch 4 batch id 1 loss 0.44234222173690796 train acc 0.84375
100%|██████████| 24/24 [00:18<00:00,  1.29it/s]epoch 4 train acc 0.8160342261904762

100%|██████████| 6/6 [00:01<00:00,  3.49it/s]epoch 4 test acc 0.8541666666666666

  4%|▍         | 1/24 [00:00<00:17,  1.32it/s]epoch 5 batch id 1 loss 0.32544636726379395 train acc 0.84375
100%|██████████| 24/24 [00:19<00:00,  1.25it/s]epoch 5 train acc 0.8815104166666666

100%|██████████| 6/6 [00:01<00:00,  3.38it/s]epoch 5 test acc 0.8229166666666666

  4%|▍         | 1/24 [00:00<00:18,  1.23it/s]epoch 6 batch id 1 loss 0.18970538675785065 train acc 0.90625
100%|██████████| 24/24 [00:19<00:00,  1.20it/s]epoch 6 train acc 0.9252232142857143

100%|██████████| 6/6 [00:01<00:00,  3.23it/s]epoch 6 test acc 0.8229166666666666

  4%|▍         | 1/24 [00:00<00:19,  1.20it/s]epoch 7 batch id 1 loss 0.06992894411087036 train acc 0.96875
100%|██████████| 24/24 [00:20<00:00,  1.19it/s]epoch 7 train acc 0.9555431547619048

100%|██████████| 6/6 [00:01<00:00,  3.31it/s]epoch 7 test acc 0.828125

  4%|▍         | 1/24 [00:00<00:18,  1.21it/s]epoch 8 batch id 1 loss 0.0820583701133728 train acc 0.96875
100%|██████████| 24/24 [00:19<00:00,  1.22it/s]epoch 8 train acc 0.9789806547619048

100%|██████████| 6/6 [00:01<00:00,  3.38it/s]epoch 8 test acc 0.8489583333333334

  4%|▍         | 1/24 [00:00<00:18,  1.22it/s]epoch 9 batch id 1 loss 0.020061591640114784 train acc 1.0
100%|██████████| 24/24 [00:19<00:00,  1.22it/s]epoch 9 train acc 0.9869791666666666

100%|██████████| 6/6 [00:01<00:00,  3.34it/s]epoch 9 test acc 0.8385416666666666

  4%|▍         | 1/24 [00:00<00:18,  1.22it/s]epoch 10 batch id 1 loss 0.016379622742533684 train acc 1.0
100%|██████████| 24/24 [00:19<00:00,  1.21it/s]epoch 10 train acc 0.9869791666666666

100%|██████████| 6/6 [00:01<00:00,  3.27it/s]epoch 10 test acc 0.8385416666666666
```

KoBert는 기존 BERT 모델의 한국어에 대한 성능 한계를 극복하기 위해 개발됨

KoBert
정확도 : 83.85%

80% 이상의 정확도로 다른 모델에 비해 뛰어난 성능을 보임.

4. 데이터 추가 후 성능 비교

데이터 양 늘린 후 다시 성능 비교
(1200개 -> 3000 개)

```
[67] # knn 모델 평가
     model = grid_clf1.best_estimator_
     model.score(x_test, y_test)
```

```
0.7207920792079208
```

```
# logistic 모델 평가
model = grid_clf2.best_estimator_
model.score(x_test, y_test)
```

```
0.805940594059406
```

```
model.evaluate(x_test, y_test)
```

```
24/24 [==============================] - 0s 5ms/step - loss: 1.2105 - accuracy: 0.7781
[1.210509181022644, 0.7780713438987732]
```

KNN
정확도 : 74.47% >> **72.08%**

유의미한 차이 없음!

Logistic Regression
정확도 : 79.68% >> **80.59%**

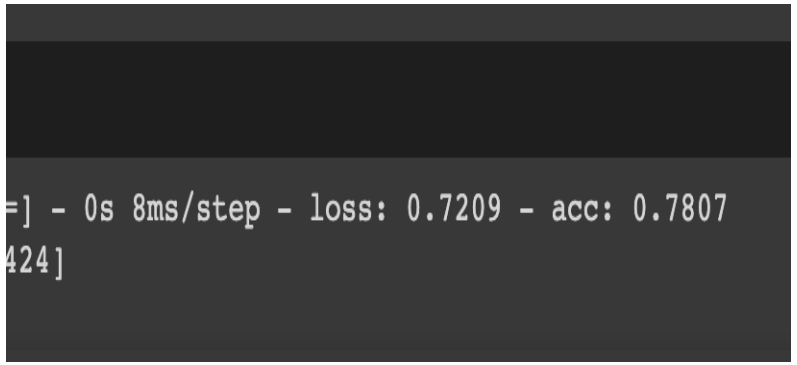
유의미한 차이 없음!

LSTM
정확도 : 77.00% >> **77.81%**

유의미한 차이 없음!

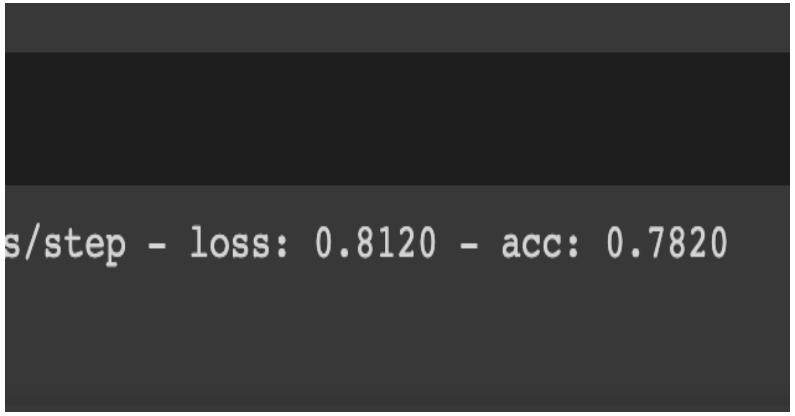
4. 데이터 추가 후 성능 비교

데이터 양 늘린 후 다시 성능 비교
(1200개 -> 3000 개)



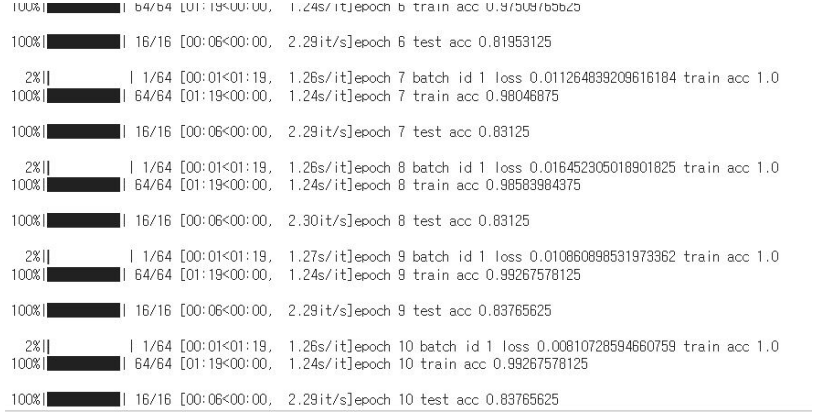
Bidirectional LSTM
정확도 : 75.26% >> **78.07%**

유의미한 차이 없음!



GRU
정확도 : 79.09% >> **78.20%**

유의미한 차이 없음!



KoBERT
정확도 : 83.85% >> **83.76%**

유의미한 차이 없음!

4. 데이터 추가 후 성능 비교

성능 높이기 위한 추가적인 노력

라벨링 데이터를 긍정/부정이 아닌 긍정/중립/부정으로 하면 더욱 정확할 것이라는 피드백 반영

기존의 라벨링
긍정 -> 1
부정 -> 0

Softmax 함수 활용

새로운 라벨링
긍정 -> 2
중립 -> 1
부정 -> 0

```
[149] model.evaluate(x_test, y_test)

9/9 [==============================] - 0s 14ms/step - loss: 1.0452 - acc: 0.7098
[1.0451939105987549, 0.7097902297973633]
```

Bidirectional LSTM
정확도 : 78.07% >> 70.9%

오히려 8% 떨어지는 결과

4. 데이터 추가 후 성능 비교

성능 높이기 위한 추가적인 노력

라벨링 데이터를 긍정/부정이 아닌 긍정/중립/부정으로 하면 더욱 정확할 것이라는 피드백 반영

기존의 라벨링
긍정 -> 1
부정 -> 0

Softmax 함수 활용

새로운 라벨링
긍정 -> 2
중립 -> 1
부정 -> 0

```
[149] model.evaluate(x_test, y_test)

9/9 [==============================] - 0s 14ms/step - loss: 1.0452 - acc: 0.7098
[1.0451939105987549, 0.7097902297973633]
```

Bidirectional LSTM
정확도 : 78.07% >> 70.9%

오히려 8% 떨어지는 결과

5. 피드백 반영한 결과 분석

피드백 1.
- **다양한 모델을 활용할 것** → **다른 모델에 비해 비교적 높은 성능을 보인 KoBert**

피드백 2.
- **데이터의 양 늘릴 것** → **유의미한 차이 없음.**

피드백 3.
- **라벨링 값 세분화** → **오히려 성능이 낮은 결과가 나옴.**

-> 해당 프로젝트에는 데이터의 양보다 모델의 설계가 더 큰 영향을 미치는 것으로 확인됨.

6. 프로젝트 기대효과

```
sentiment_predict(['음식도 맛없고 직원도 불친절해요'])
sentiment_predict(['주방장님이 엄청 친절해요'])
sentiment_predict(['서비스나 맛이 다 별로에요'])
sentiment_predict(['우리 엄마가 해준 것보다 나은듯'])
sentiment_predict(['여기 좀 심각합니다'])
sentiment_predict(['양 실화냐'])
sentiment_predict(['직원이 미친듯'])
sentiment_predict(['하.. 그냥 할말이 없음 전체적으로 왜 그 모양인지 모르겠음'])
sentiment_predict(['와 대박 부모님 모시고 싶은 맛'])
sentiment_predict(['담에 또 와야지'])
sentiment_predict(['나쁘지 않았음 다음 번에 가면 다른 메뉴 먹을 예정'])
sentiment_predict(['진짜 쓰레기'])
sentiment_predict(['개쓰레기 같은 맛 걍 장사 접어라'])
```

```
음식도 맛없고 직원도 불친절해요 -> 부정
주방장님이 엄청 친절해요 -> 긍정
서비스나 맛이 다 별로에요 -> 부정
우리 엄마가 해준 것보다 나은듯 -> 긍정
여기 좀 심각합니다 -> 긍정
양 실화냐 -> 긍정
직원이 미친듯 -> 부정
하.. 그냥 할말이 없음 전체적으로 왜 그 모양인지 모르겠음 -> 부정
와 대박 부모님 모시고 싶은 맛 -> 긍정
담에 또 와야지 -> 긍정
나쁘지 않았음 다음 번에 가면 다른 메뉴 먹을 예정 -> 긍정
진짜 쓰레기 -> 긍정
개쓰레기 같은 맛 걍 장사 접어라 -> 긍정
```

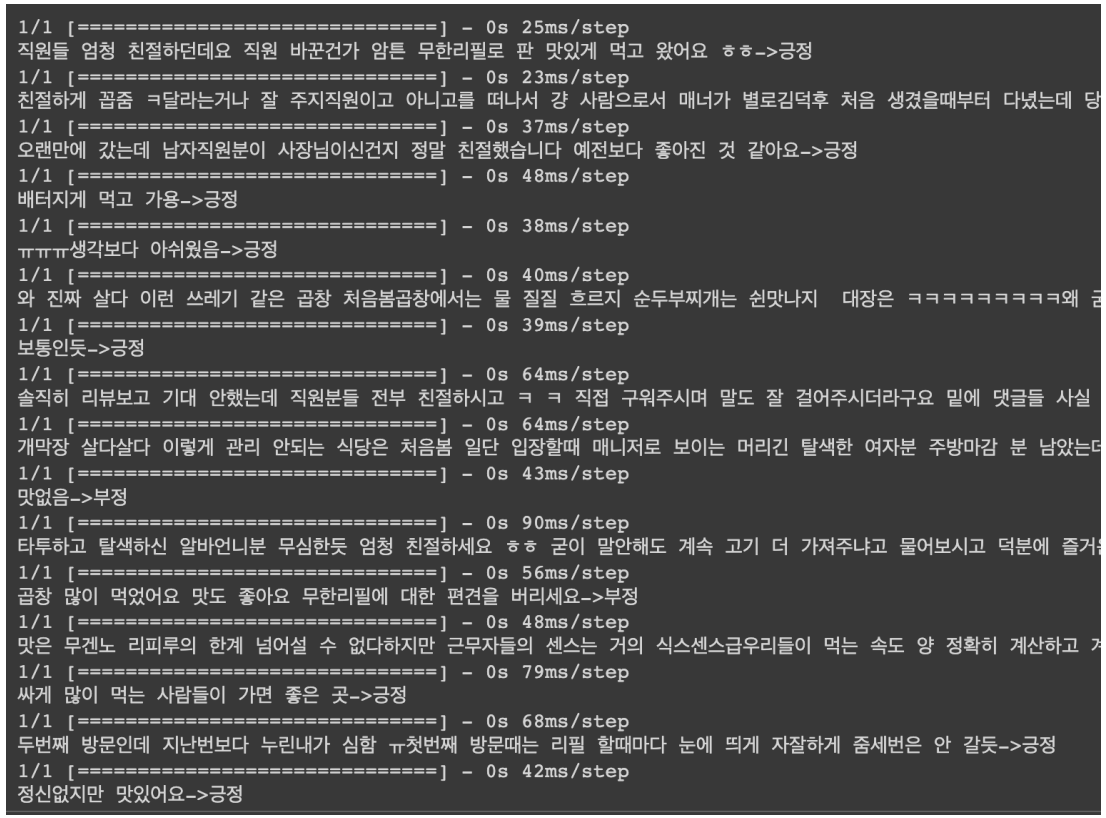
< Bidirectional LSTM을 활용한 긍/부정 평가 예시>

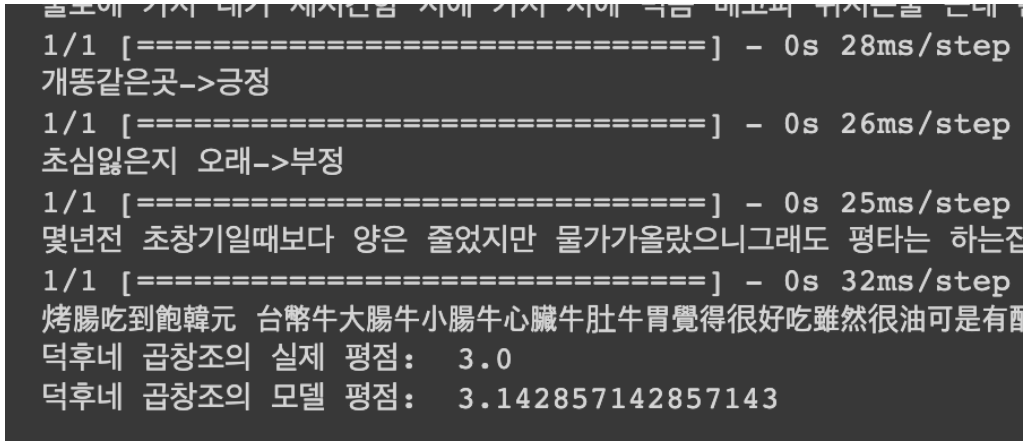
- 음식점에 대한 리뷰를 분석하는 모델을 통해 향후 홍대 맛집에 대한 새로운 평점부여 및 보다 정확한 평가가 가능

- 홍대 뿐만 아니라 다른 음식점에 대한 평가도 같은 모델을 활용하여 실시할 수 있음

6. 프로젝트 결과

- 실제 홍대 맛집에 대한 평가 진행 (상수역 인근에 위치한 "김덕후의 곱창조")





음식점 : 김덕후의 곱창조
기존 카카오맵 평점 : 3.0점
자체 모델 평점 : 3.14점

6. 프로젝트 결과

- 실제 홍대 맛집에 대한 평가 진행 (합정역 인근에 위치한 " 피오니")

```
    ->긍정
1/1 [==============================] - 0s 26ms/step
맛집맛있으나 웨이팅불편한자리그래도 맛있음->긍정
1/1 [==============================] - 0s 24ms/step
그냥그냥그런맛크림이나 딸기 모두 좀 아쉬워요->부정
1/1 [==============================] - 0s 28ms/step
딸기빙수금방녹아서 아쉬웡 ㅠㅜ->긍정
1/1 [==============================] - 0s 23ms/step
맛있네요->긍정
1/1 [==============================] - 0s 26ms/step
먹어본 딸기 케이크중 가장 맛있었음너무 달지않지도 느끼하지도 않음->긍정
1/1 [==============================] - 0s 23ms/step
봄이라 더 맛있다->긍정
1/1 [==============================] - 0s 23ms/step
딸기 생크림케익딸기 빙수->부정
피오니의 실제 평점:  3.8
피오니의 모델 평점:  3.3757961783439487
```

음식점 : 피오니
기존 카카오맵 평점 : 3.8점
자체 모델 평점 : 3.37점

-> 리뷰를 바탕으로 평가한 평점은 기존에 부여된 평점보다 낮은 것으로 확인됨

앞으로 더 많은 데이터를 확보해 다른 음식점에 대한 평가를 진행할 수 있을 것이라고 예상됨.

감사합니다