기계학습을 이용한 영화 감상평 감정 분석

최예진
2018202007
광운대학교 컴퓨터정보공학부

I. 연구 배경

A. Background

본 프로젝트는 영화 감상평 댓글 Text를 Sequence Data로 하여, Seq2Seq 기반의 모델을 만들어 영화 감상평 댓글의 감정 분석을 진행한다. 개발환경은 구글 colab으로 하며, 언어로는 python을 사용하였다.

Text를 보고 감정분석이 가능한 모델을 만들면, 영화 감상 후기의 평점에 더해 풍부한 반응 분석을 이뤄낼 수 있다.

또한, 댓글로서 반영한 사람들의 의견을 분석하여 개인별로 적용되는 영화 추천 알고리즘을 개선할 수 있고, 영화가 아닌 유튜브 등의 댓글을 input으로 하여 다양한 플랫폼에서의 활용 가능성 또한 기대할 수 있다.

B. Problem

Text를 input data로 하여 결과로 감정을 분석하는 것이 목적일 때 주의해야 할 점은, 반어법 등의 사용으로 걸러내기 어려운 noisy data가 만들어질 경우이다.

아래 사진은 영화 '클레멘타인'의 네이버 영화 평점에서 추천을 가장 많이 받아 1위를 한 댓글로, 반어법을 사용함과 동시에 평점또한 만점을 준 것을 확인할 수 있다.

부정의 의미임에도 불구하고, 사전에 영화에 대한 전체적인 배경이 없으면 정확한 감정분석이 어려운 댓글이다.

II. 연구 방법

A. Dataset / Cleaning

영화 감상평 댓글이 될 Text data는 Naver sentiment movie corpus Dataset을 사용한다. 이는 20만개의 영화 리뷰를 기록해 둔 dataset이며, Open dataset이므로 깃허브에서 다운 받을 수 있다.

해당 dataset에 포함된 모든 data는 작성자의id, 리뷰 내용, 긍부정 여부(1/0)으로 표현되어 있어 csv파일로의 변환도 가능하다.

댓글 리뷰 내용의 길이는 최대 140자이며, 긍부정 비율은 50% 내외이다. 아래 사진은 data를 출력해본 결과이다.

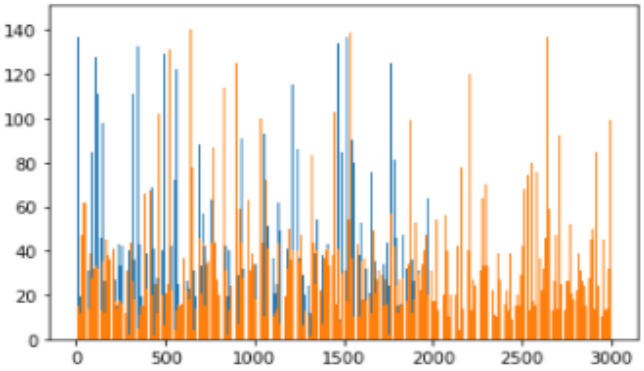
	id	document	label
0	9976970	아 더빙.. 진짜 짜증나네요 목소리	0
1	3819312	흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나	1
2	10265843	너무재밋었다그래서보는것을추천한다	0
3	9045019	교도소 이야기구먼 ..솔직히 재미는 없다..평점 조정	0
4	6483659	사이몬페그의 익살스런 연기가 돋보였던 영화!스파이더맨에서 늙어보이기만 했던 커스틴 ...	1

dataset을 cleaning하기 전에, 각 data들의 정보를 확인해보았다.

```
test_df.shape, train_df.shape
((50000, 3), (150000, 3))
```

```
id           int64
document    object
label        int64
dtype: object
```

위와 같이 test data와 train data 각각의 shape과 dtype을 알아볼 수 있다. 댓글 길이의 분포를 bar 그래프로 확인해보면 다음과 같은 형태를 볼수있다.



위의 data들을 바로 입력하지 않고 data cleaning의 전처리 과정을 거쳐가게 한다. sentencepiece라는 python 라이브러리를 설치하여 툴을 이용하였다.

```
train_df['bow'] = train_df.document.apply(lambda x: sp.encode_as_ids(str(x)))
test_df['bow'] = test_df.document.apply(lambda x: sp.encode_as_ids(str(x)))
```

```
[ ] train_df
```

	id	document	label	bow
0	9976970	아 더빙.. 진짜 짜증나네요 목소리	0	[55, 992, 5, 26, 15918, 1228]
1	3819312	흠..포스터보고 초딩영화줄...오버연기조차 가볍지 않구나	1	[1494, 6, 4134, 162, 1679, 34, 368, 50, 15251,...
2	10265843	너무재밓었다그래서보는것을추천한다	0	[18, 493, 19794, 543, 3006, 13489, 14, 2172, 292]
3	9045019	교도소 이야기구먼 ..솔직히 재미는 없다..평점 조정	0	[14221, 200, 6985, 4, 5, 3718, 949, 90, 5, 690...

첫 번째로 위와 같이 train 및 test dataset에 bow 필드를 만들어 주었다. 그런 다음 다음과 같이 train, test dataset 각각에 대해 x, y data를 생성하였다.

```
train_text = train_df.bow.values
test_text = test_df.bow.values
```

```
test_text
```

```
array([list([1351, 196]),
       list([5513, 1077, 4194, 2310, 3099, 4528, 2464, 4369, 2349, 2073, 2073, 1
       list([901, 25, 64, 479, 50, 11790, 2654, 56, 35, 5400, 13, 8961, 4834]),
       ...,
       list([2362, 11, 454, 2031, 11, 13109, 95, 6, 104, 291, 4885, 123, 2151]),
       list([442, 980, 13, 53, 643, 7, 5, 445, 156, 9885, 1808, 1171, 5, 75, 192
       list([1773, 13, 147, 3267])], dtype=object)
```

이후 생성한 x, y data를 BOW 형태로 변경한다.

```
train_bow_text = tf.keras.preprocessing.sequence.pad_sequences(train_text, value = 0)
test_bow_text = tf.keras.preprocessing.sequence.pad_sequences(test_text, value = 0)
```

```
train_bow_text
```

```
array([[    0,     0,     0, ...,    26, 15918,  1228],
       [    0,     0,     0, ...,  8263,   876,   441],
       [    0,     0,     0, ...,    14,  2172,   292],
       ...,
       [    0,     0,     0, ..., 16822,    20,    15],
       [    0,     0,     0, ...,   734,   360, 17722],
       [    0,     0,     0, ...,   304,  5016,     7]], dtype=int32)
```

불필요한 data들을 삭제하여 dataset의 크기를 줄임으로서 최종적으로 용량을 확보한다. 이를 위해 20번 이하로 나온 word를 삭제하는 과정을 거친다.

```
def cut_by_count(texts, n):
    return np.array([[word for word in text if word_count[word]>=n] for text in texts])

train_cut_text = cut_by_count(train_text, 20)
test_cut_text = cut_by_count(test_text, 20)
```

그런 다음, 위의 결과를 pad가 추가된 bow형태로 변환하여 길이를 확인해보면 아래 사진과 같다.

```
train_cut_bow_text = tf.keras.preprocessing.sequence.pad_sequences(train_cut_text, value=0)
test_cut_bow_text = tf.keras.preprocessing.sequence.pad_sequences(test_cut_text, value=0)
```

```
train_cut_bow_text.shape, test_cut_bow_text.shape
```

```
((150000, 132), (50000, 106))
```

B. Feature Manipulation

pad가 추가된 bow형태의 dataset에 길이 제한을 주어 다시 정제하면 다음과 같은 bow data를 얻는다.

```
train_cut_bow_text2 = tf.keras.preprocessing.sequence.pad_sequences(train_text, value=0, maxlen = 100)
test_cut_bow_text2 = tf.keras.preprocessing.sequence.pad_sequences(test_text, value=0, maxlen = 100)
```

```
train_cut_bow_text2.shape, test_cut_bow_text2.shape
```

```
((150000, 100), (50000, 100))
```

train 및 test dataset에 대해 각각 sentiment를 얻은 후, 각각의 sentiment를 onehot encoding으로 변경한다.

```
train_sentiment = train_df.label.values
test_sentiment = test_df.label.values
```

```
train_sentiment
```

```
array([0, 1, 0, ..., 0, 1, 0])
```

```
np.unique(train_sentiment), np.unique(test_sentiment)

train_onehot_sentiment = keras.utils.to_categorical(train_sentiment)
test_onehot_sentiment = keras.utils.to_categorical(test_sentiment)
```

onehot encoding된 sentiment의 shape는 train 및 test dataset이 각각 (150000, 2), (50000, 2)의 모양이다. 마지막으로 0과 1로 표현된 긍부정 감정 label을 text label로 변환한 뒤, 예시를 출력해보면 다음과 같다.

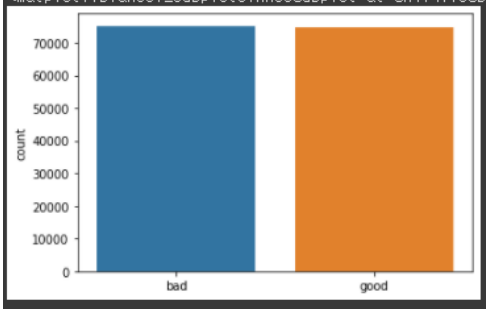
```
raw_labels=['bad', 'good']

def sentiment2label(idx):
    return raw_labels[idx]

sentiment2label(0), sentiment2label(1)
```

```
('bad', 'good')
```

label의 실제 분포를 확인해보면 다음과 같다.

```
sns.countplot(train_sentiment)
sns.countplot(list(map(lambda x:sentiment2label(x), train_sentiment)))
#plt.xticks(rotation=45)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f411c5bc9d0>
```



C. Classification / Regression

Seq2Seq의 Encoder를 이용하여 classification model 구축을 진행한다. 다음은 그에 해당하는 코드이다.

```
def Seq2Seq():
    inputs_x_bow = Input(shape=(100,))
    embedding = Embedding(20000, 120)
    x = embedding(inputs_x_bow)
    z = GRU(64)(x)
    y = Dense(2, activation='softmax')(z)

    model = Model(inputs_x_bow, y)
    model.compile(loss = 'categorical_crossentropy',
                  optimizer='adam', metrics=['accuracy'])
    return model
```

위에서 만든 함수를 이용해 학습 모델을 생성하고, 그에 대한 summary를 출력해서 확인해본 결과이다.

```
model = Seq2Seq()

model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 100)]             0
_________________________________________________________________
embedding (Embedding)        (None, 100, 120)          2400000
_________________________________________________________________
gru (GRU)                    (None, 64)                35712
_________________________________________________________________
dense (Dense)                (None, 2)                 130
=================================================================
Total params: 2,435,842
Trainable params: 2,435,842
Non-trainable params: 0
_________________________________________________________________
```

만들어놓은 bow형태의 dataset과 onehot encoding된 sentiment로 모델을 학습하는 과정은 다음과 같다.

```
hist = model.fit(train_cut_bow_text2, train_onehot_sentiment,
                 validation_data=(test_cut_bow_text2, test_onehot_sentiment),
                 verbose=1, epochs=2)
```

```
Epoch 1/2
4688/4688 [==============================] - 173s 29ms/step - loss: 0.4163 - accuracy: 0.8023
Epoch 2/2
4688/4688 [==============================] - 136s 29ms/step - loss: 0.2409 - accuracy: 0.9014
```

다음으로는 bow형태의 data들의 길이를 100으로 수정하고 이를 입력하여 모델 학습을 진행하였다.

```
train_cut_bow_text.shape
train_cut_bow_text[..., -100:][0]
```

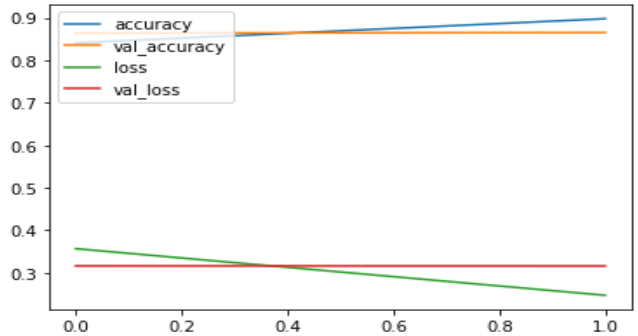
```
array([    0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
```

```
[ ]  # 학습모델을 학습해보세요.
    hist2 = model2.fit(train_cut_bow_text[..., -100:], train_onehot_sentiment,
                 validation_data=(test_cut_bow_text[..., -100:], test_onehot_sentiment),
                 verbose=1, epochs=2)
```

```
    Epoch 1/2
    4688/4688 [==============================] - 143s 30ms/step - loss: 0.4238 - accuracy: 0.7956 -
    Epoch 2/2
    4688/4688 [==============================] - 143s 31ms/step - loss: 0.2489 - accuracy: 0.8963 -
```

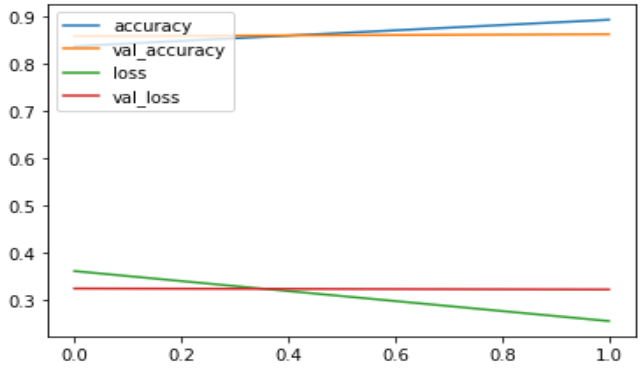
D. Evaluation

bow형태 dataset의 길이를 100으로 수정하기 이전 모델의 train accuracy, validation accuracy, train loss, validation loss를 plt를 이용하여 확인해본 결과이다.



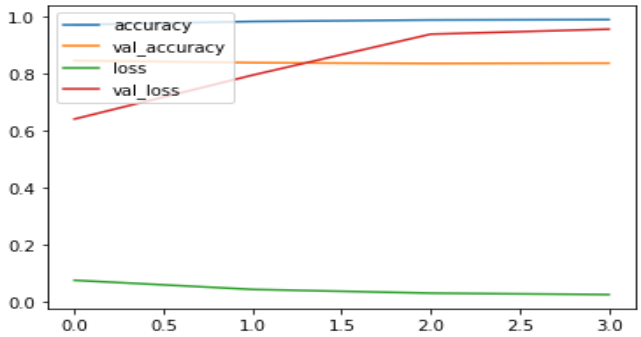
학습을 진행하는데에 약 5분의 시간이 걸렸으며, 이는 epoch를 2로 설정하였기에 크게 오랜 시간이 걸리지 않았다. 위와 같이 그래프로 확인해본 결과는 accuracy는 조금 올라가고, loss는 내려가는 것을 확인할 수 있었다.

bow형태 dataset의 길이를 100으로 수정한 이후 만든 모델의 train accuracy, validation accuracy, train loss, validation loss를 plt를 이용하여 확인해본 결과이다.



이 역시도 학습에 5분 정도의 시간이 걸렸는데, 100으로 길이를 수정후의 성능 향상이 존재하기는 했지만 아주 미미한 것을 확인하였다.

다음은 2로 설정해주었던 epochs를 2배 값인 4로 하여 모델을 만든 후 plt로 확인해 본 결과이다.



epochs를 늘리면서 모델 학습의 시간이 2배로 더 걸리게 되었고, 그만큼 정확도의 향상과 손실의 감소를 기대하였는데 예상과 다르게 loss는 매우 올라가고, accuracy는 1에 수렴해버렸다.

epochs를 크게 올리지 않았기에 큰 변화 없이 성능이 올라갈 것이라고 생각했는데, loss는 감소한 상태에서 valid loss만 크게 증가한 것을 보아 epoch만을 늘릴 경우 overfitting이 발생함을 확인하였다.

III. 연구 결과 및 고찰

A. Result

```
print(model.predict(text2bow('이 영화쏙 보세요.')[...,-100:]))
print(model.predict(text2bow('이 영화 보지 마세요.')[...,-100:]))
print(model.predict(text2bow('이 영화 보지 마세요. 나만 불거니까')[...,-100:]))

[[0.05234222 0.94765776]]
[[0.8432596 0.1567404]]
[[0.42074233 0.57925767]]
```

model test

epochs를 2로 하여 만들어 낸 모델에서 test data 10개를 예측하고, 결과를 확인해보았더니 10개의 data 모두 실제 감정에 부합하는 결과를 얻었다. 다만 마지막 평가 내용의 경우 긍/부정을 평가하기 조금 어려운 내용임에서 새로운 문제를 발견하였다.

```
res1 = model.predict(test_cut_bow_text2[0:10])
print(res1)

for i in range(0, 10):
  print(bow2text(test_text[i]))

[[0.02678623 0.97321373]
 [0.11050471 0.8894953 ]
 [0.977467   0.02253297]
 [0.9784872  0.02151277]
 [0.83897656 0.16102344]
 [0.02932082 0.9706792 ]
 [0.98661554 0.01338445]
 [0.84540886 0.15459119]
 [0.99440587 0.0055941 ]
 [0.0414608  0.9585392 ]]
굳 ㅋ
GDNTOPCLASSINTHECLUB
뭐야 이 평점들은.... 나쁘진 않지만 10점 짜리는 더더욱 아니잖아
지루하지는 않은데 완전 막장임... 돈주고 보기에는....
3D만 아니었어도 별 다섯 개 줬을텐데.. 왜 3D로 나와서 제 심기를 불편하게 하죠??
음악이 주가 된, 최고의 음악영화
진정한 쓰레기
마치 미국애니에서 튀어나온듯한 창의력없는 로봇디자인부터가,고개를 젖게한다
갈수록 개판되가는 중국영화 유치하고 내용없음 폼잡다 끝남 말도안되는 무기에 유치한cg남무
이별의 아픔뒤에 찾아오는 새로운 인연의 기쁨 But, 모든 사람이 그렇지는 않네..
```

모델 테스트를 위하여 bow형태의 data로 입력 data를 바꿔주는 함수를 만들었다. test를 원하는 문장을 넣으면 bow data로 바꿔주는 역할의 함수이다.

```
def text2bow(text, maxlen=150):
  seq = sp.encode_as_ids(text)
  bow = tf.keras.preprocessing.sequence.pad_sequences([seq], value=0, maxlen=maxlen)
  return bow
```

이후 맨 위에 첨부한 model test 사진과 같이, 긍정의 내용을 담는 text와 부정, 긍정을 차례대로 넣어보았더니 각각 긍정, 부정, 긍부정의 중간의 결과를 확인 할 수 있었다.

B. Conclusion

본 프로젝트와 같은 NLP의 경우 흥미로운 주제를 만들어 낼 dataset이 많고, 이는 다양한 분야에 활용될 가능성을 갖고있다. 비교적 dataset을 모으기 쉬우며, 전처리에 있어서도 다양한 툴이 제공됨을 알 수 있었다.

다만, 언어의 특성을 고려해보면 대부분의 툴이 영어에 집중되어있어 모든 툴이 한글을 정교하게 분석해내지는 못했다.

또한, 위에 첨부한 사진 model test에서 3번 째 입력과 같이 사람이 보았을 때는 긍정의 의미로 판단 가능한 문장이 모델의 관점에서는 중립의 평가를 하게 만든다. 이는 언어 유희, 반어법 등과 같은 표현 방법을 사용했기 때문이다.

이러한 문장들까지 구분해내려면 사람에게도 경험이 중요하듯이, 모델 학습을 시킬 때 더 많고 다양한 데이터를 입력해주어야 한다고 생각했다.

epoch를 늘리며 더 오랜 시간 모델 학습을 기다렸는데 불구하고 예상치 못한 결과값을 보았다. 프로그램 개발에 있어 시간은 매우 중요한 자원이므로 이를 낭비하지 않기 위해 모델 성능에 대한 연구를 중점적으로 해야겠다고 생각했다. 여기에는 또한 dataset의 양과 질의 중요성또한 따른다는 것을 체감하였다.

본 프로젝트에서는 Naver open dataset을 사용했지만, 크롤링을 통해 더 많은 dataset을 얻어 전처리 한 후 모델 학습에 이용하면 더 좋은 성능을 내는 모델을 만들 수 있을 거라고 생각하였다.

IV. 참고

[redacted]https://ebbnflow.tistory.com/122
[redacted]https://wikidocs.net/22650