Naver Movie Review Sentiment Analysis

-네이버 영화 리뷰 감성 분류하기

목차

1. 주제 선정

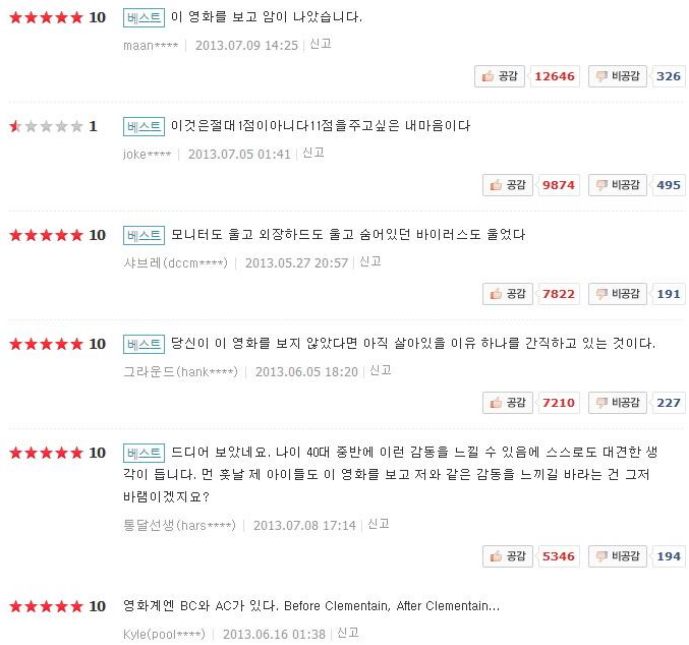
네이버 영화 리뷰 감성 분석
(Naver Movie Review Sentiment Analysis)

주제 선정 이유

1. NLP(자연어 처리)에 대한 호기심, 특히 한글 처리

2. 튜터님과의 주제 상담을 통해,
   (1) 데이터 크기 방대
   (2) 다른 사례를 바탕으로 여러 시도를 해보기 용이
   (3) 처음부터 지나치게 복잡한 데이터 다루는 것 지양

3. 다른 프로젝트에도 활용할 수 있는 여지 多



2. 데이터 전처리

(1) 데이터 개요

1. 데이터 출처
   kaggle.com/soohyun/naver-movie-review-dataset

2. 데이터 로드 및 확인

```
urllib.request.urlretrieve("https://raw.githubusercontent.com/e9t/nsmc/master/ratings_train.txt", filename="ratings_train.txt")
urllib.request.urlretrieve("https://raw.githubusercontent.com/e9t/nsmc/master/ratings_test.txt", filename="ratings_test.txt")
```

```
train_data = pd.read_table('ratings_train.txt')
test_data = pd.read_table('ratings_test.txt')
```

```
print('훈련용 리뷰 개수 :',len(train_data)) # 훈련용 리뷰 개수 출력
train_data[:5] # 상위 5개 출력
```

훈련용 리뷰 개수 : 150000

	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

```
print('테스트용 리뷰 개수 :',len(test_data)) # 테스트용 리뷰 개수 출력
test_data[:5] # 상위 5개 출력
```

테스트용 리뷰 개수 : 50000

	id	document	label
0	6270596	굳 ㅋ	1
1	9274899	GDNTOPCLASSINTHECLUB	0
2	8544678	뭐야 이 평점들은.... 나쁘진 않지만 10점 짜리는 더더욱 아니잖아	0
3	6825595	지루하지는 않은데 완전 막장임... 돈주고 보기에는....	0
4	6723715	3D만 아니었어도 별 다섯 개 줬을텐데.. 왜 3D로 나와서 제 심기를 불편하게 하죠??	0

→ train data와 test data 모두 id, document, label로 구성되어 있으며,
train data의 개수는 15만 개, test 데이터 개수는 5만 개로 확인

2. 데이터 전처리

(2) 데이터 정제

1. 중복 제거

```
train_data['document'].nunique(), train_data['label'].nunique()
train_data.drop_duplicates(subset=['document'], inplace=True) # document 열에서 중복인 내용이 있다면 중복 제거
print('총 샘플의 수 :',len(train_data))
```

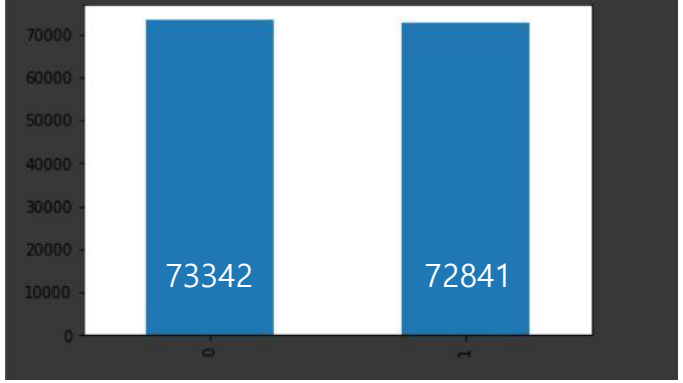
```
총 샘플의 수 : 146183
```

→ 중복을 제거한 뒤 train data의 수가 15만 개에서 146,183개로 줄어듬

이후 긍, 부정 유무가 기재되어 있는 label 값 분포 확인

```
train_data['label'].value_counts().plot(kind = 'bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7efca7d30630>
```




```
print(train_data.groupby('label').size().reset_index(name = 'count'))
```

	label	count
0	0	73342
1	1	72841

→ label 값=0인 데이터가 73,342개, label 값=1인 데이터가 72,841개로 label 값의 분포가 비교적 균일함을 알 수 있음.

2. 데이터 전처리

(2) 데이터 정제

2. 한글, 공백을 제외한 값을 제거 (정규 표현식 사용)

```
train_data['document'] = train_data['document'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣 ]","")
# 한글과 공백을 제외하고 모두 제거
train_data[:5]
```

	id	document	label
0	9976970	아 더빙 진짜 짜증나네요 목소리	0
1	3819312	흠포스터보고 초딩영화줄오버연기조차 가볍지 않구나	1
2	10265843	너무재밋었다그래서보는것을추천한다	0
3	9045019	교도소 이야기구먼 솔직히 재미는 없다평점 조정	0
4	6483659	사이몬페그의 익살스런 연기가 돋보였던 영화스파이더맨에서 늙어보이기만 했던 커스틴 던...	1

→ 한글 혹은 공백이 아닌 문자로만 구성된 review가 존재하기 때문에, 아래와 같이 Null 값을 제거함

3. Null 값 제거

```
train_data = train_data.dropna(how = 'any')# 모든행에서 Na값이 하나라도 있으면 행 삭제( .dropna(how = 'any') )

print(len(train_data))
```

```
145791
```

```
print(train_data.isnull().values.any()) # Null 값이 존재하는지 확인
```

```
False
```

→ Null 값을 제거한 결과 train data가 146,183개에서 145,791개로 줄어듬을 확인.
같은 작업을 test data에도 진행.

2. 데이터 전처리

(2) 데이터 정제

2. 한글, 공백을 제외한 값을 제거 (정규 표현식 사용)

```
train_data['document'] = train_data['document'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣 ]","")
# 한글과 공백을 제외하고 모두 제거
train_data[:5]
```

	id	document	label
0	9976970	아 더빙 진짜 짜증나네요 목소리	0
1	3819312	흠포스터보고 초딩영화줄오버연기조차 가볍지 않구나	1
2	10265843	너무재밋었다그래서보는것을추천한다	0
3	9045019	교도소 이야기구먼 솔직히 재미는 없다평점 조정	0
4	6483659	사이몬페그의 익살스런 연기가 돋보였던 영화스파이더맨에서 늙어보이기만 했던 커스틴 던...	1

→ 한글 혹은 공백이 아닌 문자로만 구성된 review가 존재하기 때문에, 아래와 같이 Null 값을 제거함

3. Null 값 제거

```
train_data = train_data.dropna(how = 'any')# 모든행에서 Na값이 하나라도 있으면 행 삭제( .dropna(how = 'any') )

print(len(train_data))
```

```
145791
```

```
print(train_data.isnull().values.any()) # Null 값이 존재하는지 확인
```

```
False
```

→ Null 값을 제거한 결과 train data가 146,183개에서 145,791개로 줄어듬을 확인.
같은 작업을 test data에도 진행.

2. 데이터 전처리

(3) 토큰화

1. 불용어 정의

```
stopwords = ['의','가','이','은','들','는','좀','잘','걍','과','도','를','으로','자','에','와','한','하다']
```

→ 한국어의 조사, 접속사 등의 보편적인 불용어를 사용 할 수 있음. 계속해서 추가 가능.

2. KoNLPy의 Okt (형태소 분석기)를 사용한 토큰화

```
X_train = []
for sentence in train_data['document']:
    temp_X = []
    temp_X = okt.morphs(sentence, stem=True) # 토큰화
    temp_X = [word for word in temp_X if not word in stopwords] # 불용어 제거
    X_train.append(temp_X)
```

2. 데이터 전처리

(4) 정수 인코딩

1. 빈도수 낮은 단어 배제를 위한 비중 확인

```
threshold = 3
total_cnt = len(tokenizer.word_index) # 단어의 수
rare_cnt = 0 # 등장 빈도수가 threshold보다 작은 단어의 개수를 카운트
total_freq = 0 # 훈련 데이터의 전체 단어 빈도수 총 합
rare_freq = 0 # 등장 빈도수가 threshold보다 작은 단어의 등장 빈도수의 총 합

# 단어와 빈도수의 쌍(pair)을 key와 value로 받는다.
for key, value in tokenizer.word_counts.items():
    total_freq = total_freq + value

    # 단어의 등장 빈도수가 threshold보다 작으면
    if(value < threshold):
        rare_cnt = rare_cnt + 1
        rare_freq = rare_freq + value

print('단어 집합(vocabulary)의 크기 :',total_cnt)
print('등장 빈도가 %s번 이하인 희귀 단어의 수: %s'%(threshold - 1, rare_cnt))
print("단어 집합에서 희귀 단어의 비율:", (rare_cnt / total_cnt)*100)
print("전체 등장 빈도에서 희귀 단어 등장 빈도 비율:", (rare_freq / total_freq)*100)
```

```
단어 집합(vocabulary)의 크기 : 43752
등장 빈도가 2번 이하인 희귀 단어의 수: 24337
단어 집합에서 희귀 단어의 비율: 55.62488571950996
전체 등장 빈도에서 희귀 단어 등장 빈도 비율: 1.8715872104872904
```

→ 등장 빈도가 2회 이하인 단어들은 단어 집합에서 무려 절반 이상을 차지하지만,
실제로 훈련 데이터에서 등장 빈도로 차지하는 비중은 상대적으로 매우 적은 수치인 1.87%밖에 되지 않음.

2. 단어 집합크기 정하기 & train, test data 정수 인코딩

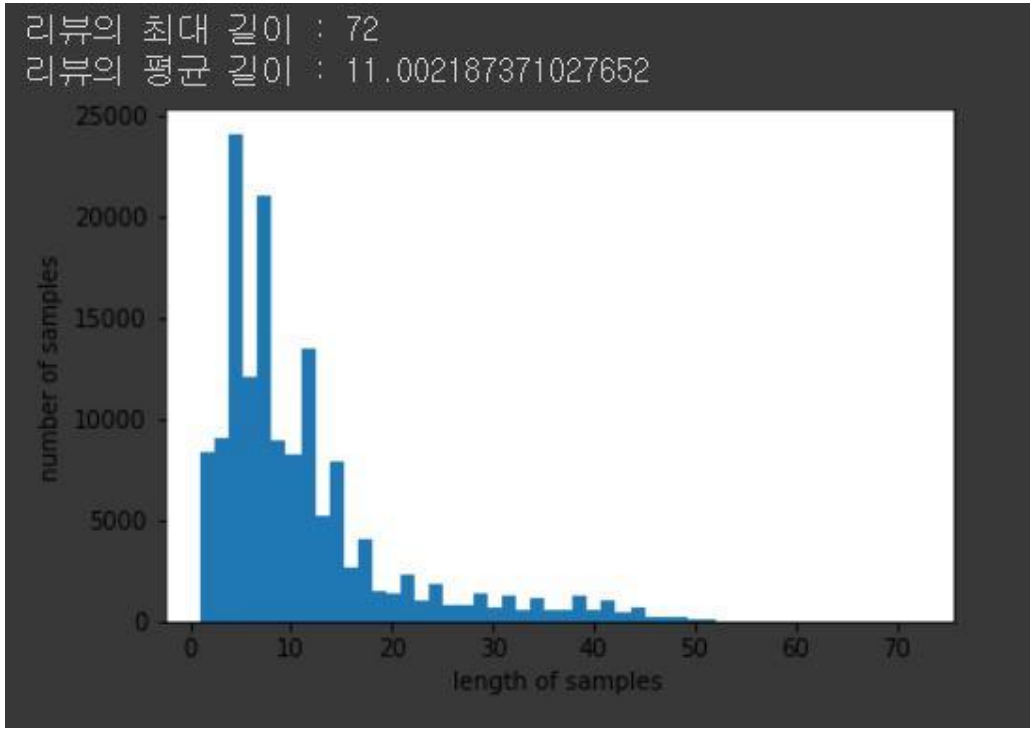
```
# 전체 단어 개수 중 빈도수 2이하인 단어 개수는 제거.
# 0번 패딩 토큰과 1번 OOV 토큰을 고려하여 +2
vocab_size = total_cnt - rare_cnt + 2
print('단어 집합의 크기 :',vocab_size)

tokenizer = Tokenizer(vocab_size, oov_token = 'OOV')
tokenizer.fit_on_texts(X_train)
X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)
```

2. 데이터 전처리

(5) 패딩

1. 전체 데이터의 리뷰 길이 파악



→ 가장 긴 리뷰의 길이는 72이며, 그래프를 봤을 때 전체 데이터의 길이 분포는 대체적으로 약 11내외의 길이

2. 데이터 전처리

(5) 패딩

2. 샘플의 길이 맞추기

```
def below_threshold_len(max_len, nested_list):
  cnt = 0
  for s in nested_list:
    if(len(s) <= max_len):
        cnt = cnt + 1
  print('전체 샘플 중 길이가 %s 이하인 샘플의 비율: %s'%(max_len, (cnt / len(nested_list))*100))
```

→ 전체 샘플 중 길이가 max_len 이하인 샘플의 비율이 몇 %인지 확인하는 함수

```
max_len = 30
below_threshold_len(max_len, X_train)
```

```
전체 샘플 중 길이가 30 이하인 샘플의 비율: 94.0830925849498
```

→ max_len = 30이 적당할 것 같음
얼마나 많은 리뷰 길이를 커버하는지 확인

전체 샘플 중 길이가 30 이하인 샘플의 비율: 94.0830925849498

```
#모든 샘플의 길이를 30으로 맞추겠습니다.

X_train = pad_sequences(X_train, maxlen = max_len)
X_test = pad_sequences(X_test, maxlen = max_len)
```

→ train, test data 길이를 모두 30으로 맞춤

3. 모델 생성 및 학습 - Word Embedding을 중심으로

1. Keras에서 제공하는 Embedding() API 외에 pretrained word embedding을 활용하여 정확성을 제고하고자 함.

2. 사용한 pretained 모델은 다음과 같음.

   (1) Word2Vec
   : 박규병님 깃허브 - https://github.com/Kyubyong/wordvectors

   (2) GloVe
   : ratsgo.github.io/embedding/tokenize.html*

   (3) FastText
   : ratsgo.github.io/embedding/tokenize.html

* 한국어 위키백과, KorQuAD, 네이버 영화 말뭉치를 은전한닢(mecab)으로 형태소 분석한 말뭉치로 학습

3. 모델 생성 및 학습 - Word Embedding을 중심으로

3. Word Embedding 기법의 차이만을 측정하기 위해 나머지 조건은 모두 다음과 같이 통일함.

```
model.add(LSTM(128, recurrent_dropout=0.1))
model.add(Dropout(0.25))
model.add(Dense(64))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
mc = ModelCheckpoint('best_model_fasttext_EMBEDDING_DIM_100.h5',
     monitor='val_acc', mode='max', verbose=1, save_best_only=True)

fasttext_model.compile(optimizer='rmsprop', loss='binary_crossentropy',
                       metrics=['acc'])
history = fasttext_model.fit(X_train, y_train, epochs=15, callbacks=[es, mc],
          batch_size=64, validation_split=0.2)
```

3. 모델 생성 및 학습 - Word Embedding을 중심으로 (1) Word2Vec

1. 코드

```
[56] from gensim.models.word2vec import Word2Vec
```

```
[58] word2vec_model = gensim.models.Word2Vec.load('./drive/My Drive/NLP_project/ko.bin')
```

```
[59] # embedding_matrix = np.zeros((VOCAB_SIZE, EMBEDDING_DIM))
     embedding_matrix = np.zeros((145791,200))
```

```
[60] # tokenizer에 있는 단어 사전을 순회하면서 word2vec의 200차원 vector를 가져옵니다

     for word, idx in tokenizer.word_index.items():
       embedding_vector = word2vec_model[word] if word in word2vec_model else None
       if embedding_vector is not None:
         embedding_matrix[idx] = embedding_vector
```

```
[144] word2vec_model = Sequential()
      word2vec_model.add(Embedding(145791,
                          200,
                          input_length=max_len,
                          weights=[embedding_matrix], # weight는 바로 위의 embedding_matrix 대입
                          trainable=False # embedding layer에 대한 train은 꼭 false로 지정
                         )
                )
      word2vec_model.add(LSTM(128, recurrent_dropout=0.1))
      word2vec_model.add(Dropout(0.25))
      word2vec_model.add(Dense(64))
      word2vec_model.add(Dropout(0.3))
      word2vec_model.add(Dense(1, activation='sigmoid'))
```

```
[113] es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
      mc = ModelCheckpoint('best_model_word2vec_EMBEDDING_DIM_200.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)
```

3. 모델 생성 및 학습 - Word Embedding을 중심으로 (1) Word2Vec

2. 결과

```
[114] word2vec_model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
      history = word2vec_model.fit(X_train, y_train, epochs=15, callbacks=[es, mc], batch_size=64, validation_split=0.2)
```

```
Epoch 1/15
1818/1818 [==============================] - ETA: 0s - loss: 0.5399 - acc: 0.7190
Epoch 00001: val_acc improved from -inf to 0.73562, saving model to best_model_word2vec_EMBEDDING_DIM_200.h5
1818/1818 [==============================] - 206s 114ms/step - loss: 0.5399 - acc: 0.7190 - val_loss: 0.5122 - val_acc: 0.7356
Epoch 2/15
1818/1818 [==============================] - ETA: 0s - loss: 0.4826 - acc: 0.7585
Epoch 00002: val_acc improved from 0.73562 to 0.74498, saving model to best_model_word2vec_EMBEDDING_DIM_200.h5
1818/1818 [==============================] - 212s 117ms/step - loss: 0.4826 - acc: 0.7585 - val_loss: 0.5037 - val_acc: 0.7450
Epoch 3/15
1818/1818 [==============================] - ETA: 0s - loss: 0.4508 - acc: 0.7784
Epoch 00003: val_acc improved from 0.74498 to 0.75664, saving model to best_model_word2vec_EMBEDDING_DIM_200.h5
1818/1818 [==============================] - 208s 114ms/step - loss: 0.4508 - acc: 0.7784 - val_loss: 0.4818 - val_acc: 0.7566
Epoch 4/15
1818/1818 [==============================] - ETA: 0s - loss: 0.4231 - acc: 0.7955
Epoch 00004: val_acc improved from 0.75664 to 0.76757, saving model to best_model_word2vec_EMBEDDING_DIM_200.h5
1818/1818 [==============================] - 210s 116ms/step - loss: 0.4231 - acc: 0.7955 - val_loss: 0.4699 - val_acc: 0.7676
Epoch 5/15
1818/1818 [==============================] - ETA: 0s - loss: 0.3986 - acc: 0.8098
Epoch 00005: val_acc improved from 0.76757 to 0.76898, saving model to best_model_word2vec_EMBEDDING_DIM_200.h5
1818/1818 [==============================] - 209s 115ms/step - loss: 0.3986 - acc: 0.8098 - val_loss: 0.4728 - val_acc: 0.7690
Epoch 6/15
1818/1818 [==============================] - ETA: 0s - loss: 0.3750 - acc: 0.8233
Epoch 00006: val_acc improved from 0.76898 to 0.76998, saving model to best_model_word2vec_EMBEDDING_DIM_200.h5
1818/1818 [==============================] - 208s 115ms/step - loss: 0.3750 - acc: 0.8233 - val_loss: 0.4757 - val_acc: 0.7700
Epoch 7/15
1818/1818 [==============================] - ETA: 0s - loss: 0.3533 - acc: 0.8350
Epoch 00007: val_acc did not improve from 0.76998
1818/1818 [==============================] - 208s 115ms/step - loss: 0.3533 - acc: 0.8350 - val_loss: 0.4959 - val_acc: 0.7691
Epoch 8/15
1818/1818 [==============================] - ETA: 0s - loss: 0.3341 - acc: 0.8448
Epoch 00008: val_acc did not improve from 0.76998
1818/1818 [==============================] - 211s 116ms/step - loss: 0.3341 - acc: 0.8448 - val_loss: 0.5246 - val_acc: 0.7652
Epoch 00008: early stopping
```

→ Word2Vec: accuracy가 0.76998로 확인

3. 모델 생성 및 학습 - Word Embedding을 중심으로 (1) Word2Vec

3. 어려웠던 점

앞서 정수인코딩, 빈 샘플 제거, 패딩을 진행한 후의 train data를 LSTM에 input으로 넣는 과정에서 에러 발생

→ 위 전처리를 진행하기 전 train data를 저장한 뒤, 저장한 train data와 다운로드 받은 pretrained word2vec를 비교하여 별도의 Matrix 생성

3. 모델 생성 및 학습 - Word Embedding을 중심으로 (2) GloVe

1. 코드

```
[90] glove_dictionary = {}
     with open("glove.txt") as f:
         for line in f:
             key_word = line.split()
             key_word, *rest = line.strip().split(" ")
             glove_dictionary[key_word] = np.array(rest, dtype=np.float64)
```

```
[93] glove_embedding_matrix = np.zeros((145791,100))
```

```
[94] for word, idx in tokenizer.word_index.items():
       glove_embedding_vector = glove_dictionary[word] if word in glove_dictionary else None
       if glove_embedding_vector is not None:
         glove_embedding_matrix[idx] = glove_embedding_vector
```

```
[106] glove_model = Sequential()
      glove_model.add(Embedding(145791,
                          100,
                          input_length=max_len,
                          weights=[glove_embedding_matrix], # weight는 바로 위의 embedding_matrix 대입
                          trainable=False # embedding layer에 대한 train은 꼭 false로 지정
                         )
                )
      glove_model.add(LSTM(128, recurrent_dropout=0.1))
      glove_model.add(Dropout(0.25))
      glove_model.add(Dense(64))
      glove_model.add(Dropout(0.3))
      glove_model.add(Dense(1, activation='sigmoid'))
```

```
[111] es2 = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
      mc2 = ModelCheckpoint('best_model_glove_EMBEDDING_DIM_100.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)
```

3. 모델 생성 및 학습 - Word Embedding을 중심으로 (2) GloVe

2. 결과

```
[112] glove_model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
      history = glove_model.fit(X_train, y_train, epochs=15, callbacks=[es2, mc2], batch_size=64, validation_split=0.2)

      Epoch 1/15
      1818/1818 [==============================] - ETA: 0s - loss: 0.5218 - acc: 0.7350
      Epoch 00001: val_acc improved from -inf to 0.75220, saving model to best_model_glove_EMBEDDING_DIM_100.h5
      1818/1818 [==============================] - 175s 96ms/step - loss: 0.5218 - acc: 0.7350 - val_loss: 0.4970 - val_acc: 0.7522
      Epoch 2/15
      1818/1818 [==============================] - ETA: 0s - loss: 0.4775 - acc: 0.7637
      Epoch 00002: val_acc improved from 0.75220 to 0.76503, saving model to best_model_glove_EMBEDDING_DIM_100.h5
      1818/1818 [==============================] - 175s 96ms/step - loss: 0.4775 - acc: 0.7637 - val_loss: 0.4783 - val_acc: 0.7650
      Epoch 3/15
      1818/1818 [==============================] - ETA: 0s - loss: 0.4542 - acc: 0.7791
      Epoch 00003: val_acc improved from 0.76503 to 0.77861, saving model to best_model_glove_EMBEDDING_DIM_100.h5
      1818/1818 [==============================] - 173s 95ms/step - loss: 0.4542 - acc: 0.7791 - val_loss: 0.4528 - val_acc: 0.7786
      Epoch 4/15
      1818/1818 [==============================] - ETA: 0s - loss: 0.4358 - acc: 0.7899
      Epoch 00004: val_acc improved from 0.77861 to 0.78529, saving model to best_model_glove_EMBEDDING_DIM_100.h5
      1818/1818 [==============================] - 173s 95ms/step - loss: 0.4358 - acc: 0.7899 - val_loss: 0.4438 - val_acc: 0.7853
      Epoch 5/15
      1818/1818 [==============================] - ETA: 0s - loss: 0.4190 - acc: 0.8015
      Epoch 00005: val_acc improved from 0.78529 to 0.78835, saving model to best_model_glove_EMBEDDING_DIM_100.h5
      1818/1818 [==============================] - 177s 97ms/step - loss: 0.4190 - acc: 0.8015 - val_loss: 0.4382 - val_acc: 0.7883
      Epoch 6/15
      1818/1818 [==============================] - ETA: 0s - loss: 0.4032 - acc: 0.8102
      Epoch 00006: val_acc did not improve from 0.78835
      1818/1818 [==============================] - 173s 95ms/step - loss: 0.4032 - acc: 0.8102 - val_loss: 0.4574 - val_acc: 0.7832
      Epoch 7/15
      1818/1818 [==============================] - ETA: 0s - loss: 0.3879 - acc: 0.8191
      Epoch 00007: val_acc did not improve from 0.78835
      1818/1818 [==============================] - 173s 95ms/step - loss: 0.3879 - acc: 0.8191 - val_loss: 0.4565 - val_acc: 0.7853
      Epoch 8/15
      1818/1818 [==============================] - ETA: 0s - loss: 0.3722 - acc: 0.8269
      Epoch 00008: val_acc improved from 0.78835 to 0.78938, saving model to best_model_glove_EMBEDDING_DIM_100.h5
      1818/1818 [==============================] - 174s 96ms/step - loss: 0.3722 - acc: 0.8269 - val_loss: 0.4773 - val_acc: 0.7894
      Epoch 9/15
      1818/1818 [==============================] - ETA: 0s - loss: 0.3589 - acc: 0.8343
      Epoch 00009: val_acc improved from 0.78938 to 0.79072, saving model to best_model_glove_EMBEDDING_DIM_100.h5
      1818/1818 [==============================] - 176s 97ms/step - loss: 0.3589 - acc: 0.8343 - val_loss: 0.4548 - val_acc: 0.7907
      Epoch 00009: early stopping
```

→ GloVe: accuracy가 0.78938로 확인
이전 0.76998 보다 조금 개선

3. 모델 생성 및 학습 - Word Embedding을 중심으로 (2) GloVe

3. 어려웠던 점

Word2Vec, FastText와 달리 pretrained 모델이 bin 파일이 아닌 txt로 제공되어
별도의 처리가 필요했음.
→ txt 파일의 각 행 맨 첫 단어를 key로, 이후 숫자를 리스트로 묶어 value로 삼는 dictionary 생성

3. 모델 생성 및 학습 - Word Embedding을 중심으로 (3) FastText

1. 코드

```
[136] from gensim.models.wrappers import FastText
      fasttext_model = FastText.load_fasttext_format('fasttext.bin')
```

```
      fasttext_matrix = np.zeros((145791,100))
```

```
[142] # tokenizer에 있는 단어 사전을 순회하면서 fasttext의 100차원 vector를 가져옵니다

      for word, idx in tokenizer.word_index.items():
        fasttext_vector = fasttext_model[word] if word in fasttext_model else None
        if fasttext_vector is not None:
          fasttext_matrix[idx] = fasttext_vector
```

```
[145] fasttext_model = Sequential()
      fasttext_model.add(Embedding(145791,
                          100,
                          input_length=max_len,
                          weights=[fasttext_matrix], # weight는 바로 위의 embedding_matrix 대입
                          trainable=False # embedding layer에 대한 train은 꼭 false로 지정
                         )
              )
      fasttext_model.add(LSTM(128, recurrent_dropout=0.1))
      fasttext_model.add(Dropout(0.25))
      fasttext_model.add(Dense(64))
      fasttext_model.add(Dropout(0.3))
      fasttext_model.add(Dense(1, activation='sigmoid'))
```

```
[146] es3 = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
      mc3 = ModelCheckpoint('best_model_fasttext_EMBEDDING_DIM_100.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)
```

3. 모델 생성 및 학습 - Word Embedding을 중심으로 (3) FastText

2. 결과

```
[148] fasttext_model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
      history = fasttext_model.fit(X_train, y_train, epochs=15, callbacks=[es3, mc3], batch_size=64, validation_split=0.2)

      Epoch 1/15
      1818/1818 [==============================] - ETA: 0s - loss: 0.5870 - acc: 0.6857
      Epoch 00001: val_acc improved from -inf to 0.72238, saving model to best_model_fasttext_EMBEDDING_DIM_100.h5
      1818/1818 [==============================] - 178s 98ms/step - loss: 0.5870 - acc: 0.6857 - val_loss: 0.5422 - val_acc: 0.7224
      Epoch 2/15
      1818/1818 [==============================] - ETA: 0s - loss: 0.5261 - acc: 0.7348
      Epoch 00002: val_acc improved from 0.72238 to 0.74938, saving model to best_model_fasttext_EMBEDDING_DIM_100.h5
      1818/1818 [==============================] - 178s 98ms/step - loss: 0.5261 - acc: 0.7348 - val_loss: 0.5036 - val_acc: 0.7494
      Epoch 9/15
      1818/1818 [==============================] - ETA: 0s - loss: 0.3906 - acc: 0.8185
      Epoch 00009: val_acc did not improve from 0.79767
      1818/1818 [==============================] - 188s 103ms/step - loss: 0.3906 - acc: 0.8185 - val_loss: 0.4432 - val_acc: 0.7908
      Epoch 10/15
      1818/1818 [==============================] - ETA: 0s - loss: 0.3783 - acc: 0.8254
      Epoch 00010: val_acc improved from 0.79767 to 0.80056, saving model to best_model_fasttext_EMBEDDING_DIM_100.h5
      1818/1818 [==============================] - 188s 104ms/step - loss: 0.3783 - acc: 0.8254 - val_loss: 0.4256 - val_acc: 0.8006
      Epoch 11/15
      1818/1818 [==============================] - ETA: 0s - loss: 0.3680 - acc: 0.8311
      Epoch 00011: val_acc did not improve from 0.80056
      1818/1818 [==============================] - 186s 102ms/step - loss: 0.3680 - acc: 0.8311 - val_loss: 0.4749 - val_acc: 0.7849
      Epoch 12/15
      1818/1818 [==============================] - ETA: 0s - loss: 0.3573 - acc: 0.8362
      Epoch 00012: val_acc did not improve from 0.80056
      1818/1818 [==============================] - 185s 102ms/step - loss: 0.3573 - acc: 0.8362 - val_loss: 0.4423 - val_acc: 0.7970
      Epoch 13/15
      1818/1818 [==============================] - ETA: 0s - loss: 0.3471 - acc: 0.8421
      Epoch 00013: val_acc did not improve from 0.80056
      1818/1818 [==============================] - 187s 103ms/step - loss: 0.3471 - acc: 0.8421 - val_loss: 0.4452 - val_acc: 0.7975
      Epoch 14/15
      1818/1818 [==============================] - ETA: 0s - loss: 0.3370 - acc: 0.8465
      Epoch 00014: val_acc improved from 0.80056 to 0.80087, saving model to best_model_fasttext_EMBEDDING_DIM_100.h5
      1818/1818 [==============================] - 187s 103ms/step - loss: 0.3370 - acc: 0.8465 - val_loss: 0.4528 - val_acc: 0.8009
      Epoch 00014: early stopping
```

→ FastText: accuracy가 0.80087로 확인
이전 0.78938 보다 조금 개선

3. 모델 생성 및 학습 - Word Embedding을 중심으로 (3) FastText

3. 어려웠던 점

Word2Vec와 유사하여 특별히 어려운 점은 없었음.

4. 평가

- Word Embedding의 종류별로 학습을 진행하고 이에 따라 개선된 결과를 도출한 점이 잘 되었다고 생각함.

- Word Embedding 이외의 요소를 다양하게 시도하지 못해 아쉬움.

감사합니다!