RNN 활용

1. RNN 이용 감성 분석

**- 데이터 불러오기**

**import** **keras**

**import** **pandas** **as** **pd**

df = pd.read\_csv('amazon\_cells\_labelled.txt', sep='**\t**', header=None)

df.head()

- 토큰화

케라스의 Tokenizer는 사이킷런의 CounterVectorizer와 비슷하게 텍스트의 토큰화를 해준다. 출력 형태가 TDM이 아닌 토큰의 리스트라는 것이 차이점이다.

tok = keras.preprocessing.text.Tokenizer()

텍스트가 있는 표의 0번째 열 df[0]을 바탕으로 텍스트에 나타난 단어 종류를 학습한다.

tok.fit\_on\_texts(df[0])

학습이라고는 하지만 실제로는 단어마다 고유 번호를 붙인 것이다. 예를 들어 'plug'의 단어 번호는 155번이다.

tok.word\_index['plug']

155

단어: 번호 형태의 사전은 index\_word이다.

tok.index\_word[155]

'plug'

이제 df[0]을 실제로 토큰화를 하고, 텍스트를 단어 번호의 리스트로 변환한다.

CountVectorizer와 달리 형태소 분석기를 붙이는 것이 불가능하므로 **한국어를 처리할 경우에는 미리 형태소 단위로** 띄어쓰기를 해둬야 한다.

seq = tok.texts\_to\_sequences(df[0])

첫번째 글은 다음과 같은 리스트로 바뀌어 있다. TDM에서는 33번째 단어가 1개 출현했다는 의미로 33번째 열의 값이 1이 되지만, 여기서는 단순히 33번이 맨 처음에 한 번 나온다.

seq[0]

array([ 33, 117, 5, 53, 214, 11, 47, 8, 155, 4, 19, 337, 19,

1, 546, 416, 2, 241, 190, 6, 812])

단어 번호를 역으로 단어로 바꾸려면 다음과 같이 한다.

' '.join(tok.index\_word[i] **for** i **in** seq[0])

'so there is no way for me to plug it in here in the us unless i go by a converter'

- 패딩을 해보자

먼저 텍스트의 최대 길이를 구한다.

MAXLEN = max(len(s) **for** s **in** seq)

**패딩을 통해 길이를 맞춘다**.

텍스트가 너무 긴 경우에는 최대 길이보다 더 짧은 길이로 맞추기도 한다. 그렇게 하면 너무 긴 텍스트는 일부가 잘려 나가게 된다.

pad = keras.preprocessing.sequence.pad\_sequences(seq, MAXLEN)

pad[0]

- 데이터 분할

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(pad, df[1], test\_size=.2, random\_state=1234)

- 모형 만들기

모형에서 학습할 단어의 갯수는 실제 단어의 갯수에 1을 더한다. 패딩에 사용한 0이 포함되기 때문이다.

NUM\_WORDS = len(tok.index\_word) + 1

**from** **keras.models** **import** Sequential

**from** **keras.layers** **import** Dense, Embedding, LSTM

rnn = Sequential()

먼저 임베딩 레이어를 넣어준다. 임베딩 레이어는 입력된 단어 번호를 벡터(좌표)로 바꿔준다.

아래의 예에서는 8차원의 벡터로 변환한다.

현재의 단어 번호는 단어의 의미와 상관없이 붙은 것이므로 분석에 별로 도움이 되지 않는다. 예를 들어 'phone'은 9번인데, 'smartphone'은 1657번이다. 이 두 단어는 의미가 비슷해서 서로 바꿔쓸 수도 있기 때문에 만약 비슷한 벡터를 가지게 된다면 분석에 도움이 될 것이다.

mask\_zero=True는 0으로 패딩된 값을 마스킹하여 네트워크의 뒤로 전달되지 않게 만든다. 이렇게 하면 인위적으로 패딩된 부분은 학습에 영향을 미치지 않는다.

rnn.add(Embedding(input\_dim=NUM\_WORDS, output\_dim=8, input\_length=MAXLEN, mask\_zero=True))

다음으로 LSTM 레이어를 추가한다.

아래 LSTM은 16개의 노드를 가진다. return\_sequences=False는 가장 마지막 토큰에만 다음 레이어로 출력을 내보내라는 뜻이다.

rnn.add(LSTM(16, return\_sequences=False))

이제 LSTM의 출력을 바탕으로 긍부정을 예측는 Dense 레이어를 덧붙인다.

rnn.add(Dense(1, activation='sigmoid'))

모형의 요약을 보자. NUM\_WORDS가 1,879개이고 이들 각각이 8차원 벡터로 바뀌기 때문에 임베딩 레이어는 1,879 \* 8 = 15,032개의 파라미터를 갖는다.

LSTM은 이들 8차원 벡터를 16개의 노드로 바꾼다. 여기에 파라미터가 (8+1)×16=144(8+1)×16=144개가 들어간다(+1은 바이어스다). 또한 16개의 노드에서 다시 16개의 노드로 순환되는 부분이 있기 때문에 16×16=25616×16=256개가 추가되서 400개가 된다. 그리고 이런 게이트가 4개(입력, 출력, 망각, 활성화) 있기 때문에 총 1,600개의 파라미터를 갖는다.

여기까지보면 텍스트의 전체 길이 30은 파라미터의 수에 영향을 주지 않는 것을 알 수 있다. 왜냐하면 순환신경망이기 때문에 동일한 파라미터를 갖는 구조가 30번 반복 적용될 뿐이기 때문이다.

rnn.summary()

훈련시키는 방법은 다른 신경망과 동일하다.

**from** **keras.optimizers** **import** Adam

rnn.compile(optimizer=Adam(), loss='binary\_crossentropy', metrics=['accuracy'])

rnn.fit(X\_train, y\_train, epochs=10)

- 예측

y\_rnn = rnn.predict\_classes(X\_test)

**from** **sklearn.metrics** **import** accuracy\_score

accuracy\_score(y\_test, y\_rnn)

- 역방향

단어처리 순서를 앞에서 뒤로 하는 것이 아니라 역방향으로 뒤에서 앞으로도 할 수 있다. 순환신경망 레이어에 go\_backwards=True를 추가해주면 된다.

rnn = Sequential()

rnn.add(Embedding(input\_dim=NUM\_WORDS, output\_dim=8, input\_length=MAXLEN, mask\_zero=True))

rnn.add(LSTM(16, return\_sequences=False, go\_backwards=True))

rnn.add(Dense(1, activation='sigmoid'))

- 양방향

순방향 RNN과 양방향 RNN을 합치면 양방향 RNN이 된다. Bidirectional을 사용한다.

**from** **keras.layers** **import** Bidirectional

rnn = Sequential()

rnn.add(Embedding(input\_dim=NUM\_WORDS, output\_dim=8, input\_length=MAXLEN, mask\_zero=True))

rnn.add(Bidirectional(LSTM(16, return\_sequences=False)))

rnn.add(Dense(1, activation='sigmoid'))

**2. 스펨 메일 분류**

- 데이터 다운로드

**import** pandas **as** pd

**import** urllib.request

urllib.request.urlretrieve("https://raw.githubusercontent.com/mohitgupta-omg/Kaggle-SMS-Spam-Collection-Dataset-/master/spam.csv", filename="spam.csv")

data = pd.**read**\_csv('spam.csv', encoding='latin1')

다운로드 받은 spam.csv 파일을 Pandas를 이용하여 data에 저장합니다. 총 샘플의 수를 확인해봅시다.

**print**('총 샘플의 수 :',len(data))

data.head()

레이블과 메일 내용이 담긴 v1열과 v2열만 필요하므로, Unnamed: 2, Unnamed: 3, Unnamed: 4 열은 삭제합니다. 또한, v1열에 있는 ham과 spam 레이블을 각각 숫자 0과 1로 바꾸겠습니다. 다시 data에서 5개의 행만 출력해보겠습니다.

**del** data['Unnamed: 2']

**del** data['Unnamed: 3']

**del** data['Unnamed: 4']

data['v1'] = data['v1'].replace(['ham','spam'],[0,1])

data[:5]

v1열은 정수형, v2열은 문자열 데이터를 갖고있습니다.

혹시 Null 값을 가진 샘플이 있는지 isnull().values.any()로도 확인 가능합니다.

**print**('결측값 여부 :',data.isnull().**values**.any())

결측값 여부 : **False**

중복이 있지는 않은지 확인해보겠습니다.

**print**('v2열의 유니크한 값 :',data['v2'].nunique())

총 5,572개의 샘플이 존재하는데 v2열에서 중복을 제거한 샘플의 개수가 5,169개라는 것은 403개의 중복 샘플이 존재한다는 의미입니다.

# v2 열에서 중복인 내용이 있다면 중복 제거

data.drop\_duplicates(subset=['v2'], inplace=**True**)

총 샘플의 수가 5,572개에서 5,169개로 줄었습니다. 이제 스팸 메일 유무를 의미하는 레이블 값의 분포를 시각화해보겠습니다.

data['v1'].value\_counts().plot(kind='bar')

레이블이 대부분 **0에 편중되어있는데**, 이는 스팸 메일 데이터의 대부분의 메일이 정상 메일임을 의미합니다. **수치로 확인해보겠습니다**.

**print**('정상 메일과 스팸 메일의 개수')

**print**(data.groupby('v1').size().reset\_index(name='count'))

훈련 데이터와 테스트 데이터 **모두 정상 메일은 87**%, 스팸 **메일은 12%**가 존재합니다.

이제 케라스 토크나이저를 통해 훈련 데이터에 대해서 **토큰화와 정수 인코딩** 과정을 수행해보겠습니다.

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(X\_train)

X\_train\_encoded = tokenizer.texts\_to\_sequences(X\_train)

X\_train\_encoded에는 X\_train의 각 **단어들이 맵핑되는 정수로 인코딩되어 저장되었습니다**. **5개의 메일만 출력해서 확인해보겠습니다.**

**print**(**X\_train\_encoded**[:5])

**max길치 찾기**

max(len(l) **for** l **in** X\_train\_encoded)

**가장 긴 메일의 길이는 189, 패딩 하기**

max\_len = 189

X\_train\_padded = pad\_sequences(X\_train\_encoded, maxlen = max\_len)

X\_train\_padded.shape

패딩을 위한 토큰인 0번 단어를 고려하며 +1을 해서 저장해주어야 한다는 점입니다.

vocab\_size = len(word\_to\_index) + 1

이제 LSTM을 이용하여 스팸 메일 분류기를 만들어보도록 하겠습니다.

**from** tensorflow.keras.layers **import** LSTM, Embedding, Dense

**from** tensorflow.keras.models **import** Sequential

model = Sequential()

model.add(Embedding(vocab\_size, 32)) # 임베딩 벡터의 차원은 32

model.add(LSTM(32)) # RNN 셀의 hidden\_size는 32

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary\_crossentropy', metrics=['acc'])

history = model.fit(X\_train\_padded, y\_train, epochs=4, batch\_size=64, validation\_split=0.2)

이제 테스트 데이터에 **대해서 정확도를 확인해보겠습니다**.

X\_test\_encoded = tokenizer.texts\_to\_sequences(X\_test)

X\_test\_padded = pad\_sequences(X\_test\_encoded, maxlen = max\_len)

**print**("\n 테스트 정확도: %.4f" % (model.evaluate(X\_test\_padded, y\_test)[1]))

텐스보드 추가하세요.