

- 🚀 week9 과제는 **9주차의 LSTM Model 및 GRU Model 실습**으로 구성되어 있습니다.
- 🚀 위키독스의 딥러닝을 이용한 자연어 처리 입문 교재 실습 등의 문서 자료로 구성되어 있는 과제입니다.
- 🚀 안내된 링크에 맞추어 **직접 코드를 따라 치면서 (필사)** 해당 nlp task 의 기본적인 라이브러리와 메서드를 숙지해보시면 좋을 것 같습니다 😊 필수라고 체크한 부분은 과제에 반드시 포함시켜 주시고, 선택으로 체크한 부분은 자율적으로 스터디 하시면 됩니다.
- 🚀 궁금한 사항은 깃허브 이슈나, 카톡방, 세션 발표 시작 이전 시간 등을 활용하여 자유롭게 공유해주세요!

```
import nltk
# nltk colab 환경에서 실행시 필요한 코드입니다.
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data] /root/nltk_data...
[nltk_data] Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data] /root/nltk_data...
[nltk_data] Unzipping chunkers/maxent_ne_chunker.zip.
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data] Unzipping corpora/words.zip.
True
```

## ▼ 1 LSTM Model

### 👁️ 내용 복습

- LSTM Model은 기존의 RNN이 비교적 짧은 sequence에 대해서만 효과를 보이던 단점을 극복하여 긴 sequence의 입력을 처리하는데 탁월한 성능을 보입니다.
- LSTM Model 설명 참고 자료
  - [장단기 메모리\(Long Short-Term Memory, LSTM\)](#)
  - [Understanding LSTM Networks](#)
  - [RNN과 LSTM을 이해해보자!](#)

😬 1-(1)~(4)의 실습 중 2개 이상 택해서 실습하시면 됩니다.

### ◆ 1-(1) 텍스트 생성하기

- 문맥을 반영하여 텍스트를 생성하는 모델을 만들어 보겠습니다.

📌 [LSTM을 이용한 텍스트 생성](#) 📌 1. RNN을 이용하여 텍스트 생성하기는 안 하셔도 무방합니다.

```
import pandas as pd
import numpy as np
from string import punctuation

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
```

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/week9/archive/ArticlesApril2018.csv')
df.head()
```

articleID	articleWordCount	byline	documentType	headline
-----------	------------------	--------	--------------	----------

By: JOHN

Former N.F.  
Cheerleader

```
print('열의 개수: ', len(df.columns))
print(df.columns)
```

열의 개수: 15

```
Index(['articleID', 'articleWordCount', 'byline', 'documentType', 'headline',
      'keywords', 'multimedia', 'newDesk', 'printPage', 'pubDate',
      'sectionName', 'snippet', 'source', 'typeOfMaterial', 'webURL'],
      dtype='object')
```

Effect: Less

```
print(df['headline'].isnull().values.any())
```

False

2 5adf4626068401528a2aa628

2427

The New Noma

article

Nom

```
headline = []
# 헤드라인의 값들을 리스트로 저장
headline.extend(list(df.headline.values))
headline[:5]
```

```
['Former N.F.L. Cheerleaders' Settlement Offer: $1 and a Meeting With Goodell',
 'E.P.A. to Unveil a New Rule. Its Effect: Less Science in Policymaking.',
 'The New Noma, Explained',
 'Unknown',
 'Unknown']
```

ALISTEN and

```
print('총 샘플의 개수 : {}'.format(len(headline)))
```

총 샘플의 개수 : 1324

```
headline = [word for word in headline if word != "Unknown"]
print('노이즈값 제거 후 샘플의 개수 : {}'.format(len(headline)))
```

노이즈값 제거 후 샘플의 개수 : 1214

headline[:5]

```
['Former N.F.L. Cheerleaders' Settlement Offer: $1 and a Meeting With Goodell',
 'E.P.A. to Unveil a New Rule. Its Effect: Less Science in Policymaking.',
 'The New Noma, Explained',
 'How a Bag of Texas Dirt Became a Times Tradition',
 'Is School a Place for Self-Expression?']
```

```
def reprocessing(raw_sentence):
    preproceed_sentence = raw_sentence.encode("utf8").decode("ascii", 'ignore')
    # 구두점 제거와 동시에 소문자화
    return ''.join(word for word in preproceed_sentence if word not in punctuation).lower()
```

```
preprocessed_headline = [repreprocessing(x) for x in headline]
preprocessed_headline[:5]
```

```
['former nfl cheerleaders settlement offer 1 and a meeting with goodell',
 'epa to unveil a new rule its effect less science in policymaking',
 'the new noma explained',
 'how a bag of texas dirt became a times tradition',
 'is school a place for selfexpression']
```

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(preprocessed_headline)
vocab_size = len(tokenizer.word_index) + 1
print('단어 집합의 크기 : %d' % vocab_size)
```

단어 집합의 크기 : 3494

```
sequences = list()
```

```
for sentence in preprocessed_headline:
```

```
    # 각 샘플에 대한 정수 인코딩
    encoded = tokenizer.texts_to_sequences([sentence])[0]
    for i in range(1, len(encoded)):
        sequence = encoded[:i+1]
        sequences.append(sequence)
```

```
sequences[:11]
```

```
[[99, 269],
 [99, 269, 371],
 [99, 269, 371, 1115],
 [99, 269, 371, 1115, 582],
 [99, 269, 371, 1115, 582, 52],
 [99, 269, 371, 1115, 582, 52, 7],
 [99, 269, 371, 1115, 582, 52, 7, 2],
 [99, 269, 371, 1115, 582, 52, 7, 2, 372],
 [99, 269, 371, 1115, 582, 52, 7, 2, 372, 10],
 [99, 269, 371, 1115, 582, 52, 7, 2, 372, 10, 1116],
 [100, 3]]
```

```
index_to_word = {}
```

```
for key, value in tokenizer.word_index.items(): # 인덱스를 단어로 바꾸기 위해 index_to_word를 생성
    index_to_word[value] = key
```

```
print('빈도수 상위 582번 단어 : {}'.format(index_to_word[582]))
```

```
index_to_word = {}
```

```
for key, value in tokenizer.word_index.items(): # 인덱스를 단어로 바꾸기 위해 index_to_word를 생성
    index_to_word[value] = key
```

```
print('빈도수 상위 582번 단어 : {}'.format(index_to_word[582]))
```

빈도수 상위 582번 단어 : offer

```
max_len = max(len(l) for l in sequences)
print('샘플의 최대 길이 : {}'.format(max_len))
```

샘플의 최대 길이 : 24

```
sequences = pad_sequences(sequences, maxlen=max_len, padding='pre')
print(sequences[:3])
```

```
[[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  99 269]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  99 269 371]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  99 269 371 1115]]
```

```
sequences = np.array(sequences)
X = sequences[:, :-1]
y = sequences[:, -1]
print(X[:3])
print(y[:3])
```

```
[[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0 99]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0 99 269]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0 99 269 371]
 [269 371 1115]]
```

```
y = to_categorical(y, num_classes=vocab_size)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Dense, LSTM
```

```
embedding_dim = 10
hidden_units = 128

model = Sequential()
model.add(Embedding(vocab_size, embedding_dim))
model.add(LSTM(hidden_units))
model.add(Dense(vocab_size, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, y, epochs=200, verbose=2)
```

```
244/244 - 8s - loss: 0.2796 - accuracy: 0.9164 - 8s/epoch - 32ms/step
Epoch 173/200
244/244 - 11s - loss: 0.2777 - accuracy: 0.9153 - 11s/epoch - 44ms/step
Epoch 174/200
244/244 - 9s - loss: 0.2751 - accuracy: 0.9154 - 9s/epoch - 36ms/step
Epoch 175/200
244/244 - 8s - loss: 0.2744 - accuracy: 0.9161 - 8s/epoch - 31ms/step
Epoch 176/200
244/244 - 9s - loss: 0.2732 - accuracy: 0.9166 - 9s/epoch - 39ms/step
Epoch 177/200
244/244 - 10s - loss: 0.2726 - accuracy: 0.9177 - 10s/epoch - 40ms/step
Epoch 178/200
```

```
Epoch 179/200
244/244 - 10s - loss: 0.2714 - accuracy: 0.9176 - 10s/epoch - 40ms/step
Epoch 180/200
244/244 - 9s - loss: 0.2715 - accuracy: 0.9179 - 9s/epoch - 38ms/step
Epoch 181/200
244/244 - 9s - loss: 0.2714 - accuracy: 0.9166 - 9s/epoch - 38ms/step
Epoch 182/200
244/244 - 10s - loss: 0.2710 - accuracy: 0.9166 - 10s/epoch - 39ms/step
Epoch 183/200
244/244 - 9s - loss: 0.2708 - accuracy: 0.9173 - 9s/epoch - 37ms/step
Epoch 184/200
244/244 - 12s - loss: 0.2804 - accuracy: 0.9138 - 12s/epoch - 50ms/step
Epoch 185/200
244/244 - 11s - loss: 0.2790 - accuracy: 0.9166 - 11s/epoch - 44ms/step
Epoch 186/200
244/244 - 11s - loss: 0.2686 - accuracy: 0.9184 - 11s/epoch - 44ms/step
Epoch 187/200
244/244 - 10s - loss: 0.2672 - accuracy: 0.9164 - 10s/epoch - 40ms/step
Epoch 188/200
244/244 - 10s - loss: 0.2667 - accuracy: 0.9166 - 10s/epoch - 40ms/step
Epoch 189/200
244/244 - 10s - loss: 0.2669 - accuracy: 0.9152 - 10s/epoch - 42ms/step
Epoch 190/200
244/244 - 9s - loss: 0.2739 - accuracy: 0.9152 - 9s/epoch - 39ms/step
Epoch 191/200
244/244 - 10s - loss: 0.2699 - accuracy: 0.9158 - 10s/epoch - 39ms/step
Epoch 192/200
244/244 - 10s - loss: 0.2662 - accuracy: 0.9168 - 10s/epoch - 39ms/step
Epoch 193/200
244/244 - 10s - loss: 0.2653 - accuracy: 0.9166 - 10s/epoch - 40ms/step
Epoch 194/200
244/244 - 9s - loss: 0.2645 - accuracy: 0.9157 - 9s/epoch - 37ms/step
Epoch 195/200
244/244 - 11s - loss: 0.2650 - accuracy: 0.9140 - 11s/epoch - 47ms/step
Epoch 196/200
244/244 - 11s - loss: 0.2647 - accuracy: 0.9166 - 11s/epoch - 44ms/step
Epoch 197/200
244/244 - 9s - loss: 0.2639 - accuracy: 0.9164 - 9s/epoch - 39ms/step
Epoch 198/200
244/244 - 11s - loss: 0.2632 - accuracy: 0.9167 - 11s/epoch - 45ms/step
Epoch 199/200
244/244 - 10s - loss: 0.2630 - accuracy: 0.9166 - 10s/epoch - 41ms/step
Epoch 200/200
244/244 - 10s - loss: 0.2636 - accuracy: 0.9153 - 10s/epoch - 40ms/step
244/244 - 10s - loss: 0.2660 - accuracy: 0.9150 - 10s/epoch - 42ms/step
<keras.callbacks.History at 0x7f324ac43790>
```

```
def sentence_generation(model, tokenizer, current_word, n): # 모델, 토큰라이저, 현재 단어, 반복할 횟수
    init_word = current_word
    sentence = ''

    # n번 반복
    for _ in range(n):
        encoded = tokenizer.texts_to_sequences([current_word])[0]
        encoded = pad_sequences([encoded], maxlen=max_len-1, padding='pre')

        # 입력한 X(현재 단어)에 대해서 y를 예측하고 y(예측한 단어)를 result에 저장.
```

```

result = model.predict(encoded, verbose=0)
result = np.argmax(result, axis=1)

for word, index in tokenizer.word_index.items():
    # 만약 예측한 단어와 인덱스와 동일한 단어가 있다면
    if index == result:
        break

# 현재 단어 + ' ' + 예측 단어를 현재 단어로 변경
current_word = current_word + ' ' + word

# 예측 단어를 문장에 저장
sentence = sentence + ' ' + word

sentence = init_word + sentence
return sentence

```

```

print(sentence_generation(model, tokenizer, 'i', 10))
print(sentence_generation(model, tokenizer, 'how', 10))

```

i disapprove of school vouchers can i still apply for them  
how to make a crossword puzzle industrys biggest turning point yet

### ◆ 1-(2) 로이터 뉴스 분류하기

- 로이터 뉴스 데이터를 LSTM을 이용하여 텍스트 분류를 진행하겠습니다.

🔗 [LSTM을 이용한 로이터 뉴스 분류](#) 🙌 1. 로이터 뉴스 데이터에 대한 이해는 안 하셔도 무방합니다.

### ◆ 1-(3) 네이버 영화 리뷰 감성 분류하기

- 네이버 영화 리뷰를 감성에 따라 분류하고, 임의의 리뷰에 대해 감정을 예측해보겠습니다.

🔗 [LSTM으로 네이버 영화 리뷰 감성 분류](#)

```

%%bash
apt-get update
apt-get install g++ openjdk-8-jdk python-dev python3-dev
pip3 install JPype1
pip3 install konlpy

unpacking openjdk-8-jdk-headless:amd64 (8u312-b07-0ubuntu1~18.04) ...
Selecting previously unselected package openjdk-8-jdk:amd64.
Preparing to unpack .../14-openjdk-8-jdk_8u312-b07-0ubuntu1~18.04_amd64.deb ...
Unpacking openjdk-8-jdk:amd64 (8u312-b07-0ubuntu1~18.04) ...
Setting up libgtk2.0-common (2.24.32-1ubuntu1) ...
Setting up fonts-dejavu-core (2.37-1) ...
Setting up libxxf86dga1:amd64 (2:1.1.4-1) ...
Setting up fonts-dejavu-extra (2.37-1) ...
Setting up openjdk-8-jre-headless:amd64 (8u312-b07-0ubuntu1~18.04) ...
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/orbd to provide /usr/

```

```

update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/servertool to provide
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/tnameserv to provide
Setting up libgtk2.0-0:amd64 (2.24.32-1ubuntu1) ...
Setting up libgail18:amd64 (2.24.32-1ubuntu1) ...
Setting up openjdk-8-jdk-headless:amd64 (8u312-b07-0ubuntu1~18.04) ...
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/idlj to provide /usr/bin/
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/wsimport to provide /usr/
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/jsadebugd to provide /usr
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/native2ascii to provide /
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/javah to provide /usr/bin
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/hsdb to provide /usr/bin/
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/clhsdb to provide /usr/bi
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/xjc to provide /usr/bin/x
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/schemagen to provide /usr
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/extcheck to provide /usr/
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/jhat to provide /usr/bin/
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/wsgen to provide /usr/bin
Setting up x11-utils (7.7+3build1) ...
Setting up libgail-common:amd64 (2.24.32-1ubuntu1) ...
Setting up libatk-wrapper-java (0.33.3-20ubuntu0.1) ...
Setting up libgtk2.0-bin (2.24.32-1ubuntu1) ...
Setting up libatk-wrapper-java-jni:amd64 (0.33.3-20ubuntu0.1) ...
Setting up openjdk-8-jre:amd64 (8u312-b07-0ubuntu1~18.04) ...
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/policytool to provide
Setting up openjdk-8-jdk:amd64 (8u312-b07-0ubuntu1~18.04) ...
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/appletviewer to provide /
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/jconsole to provide /usr/
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for hicolor-icon-theme (0.17-2) ...
Processing triggers for fontconfig (2.12.6-0ubuntu2) ...
Processing triggers for mime-support (3.60ubuntu1) ...
Processing triggers for libc-bin (2.27-3ubuntu1.3) ...
/sbin/ldconfig.real: /usr/local/lib/python3.7/dist-packages/ideep4py/lib/libmkldnn.so.0 is

Collecting JPype1
  Downloading JPype1-1.3.0-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.whl (448 kB)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages
Installing collected packages: JPype1
Successfully installed JPype1-1.3.0
Collecting konlpy
  Downloading konlpy-0.6.0-py2.py3-none-any.whl (19.4 MB)
Requirement already satisfied: lxml>=4.1.0 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: JPype1>=0.7.0 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: numpy>=1.6 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages
Installing collected packages: konlpy
Successfully installed konlpy-0.6.0

```

```
%env JAVA_HOME "/usr/lib/jvm/java-8-openjdk-amd64"
```

```
env: JAVA_HOME="/usr/lib/jvm/java-8-openjdk-amd64"
```

```

import konlpy
from konlpy.tag import Kkma, Komoran, Hannanum, Okt
from konlpy.utils import pprint

```



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
import urllib.request
from konlpy.tag import Okt
from tqdm import tqdm
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

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

```
('ratings_test.txt', <http.client.HTTPMessage at 0x7f3246162b50>)
```

```
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
```

```
훈련용 리뷰 개수 : 150000
```

```
train_data[:5] # 상위 5개 출력
```

	id	document	label
0	9976970	아 더빙.. 진짜 짜증나네요 목소리	0
1	3819312	흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나	1
2	10265843	너무재밌었다그래서보는것을추천한다	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
```

```
테스트용 리뷰 개수 : 50000
```

```
test_data[:5]
```

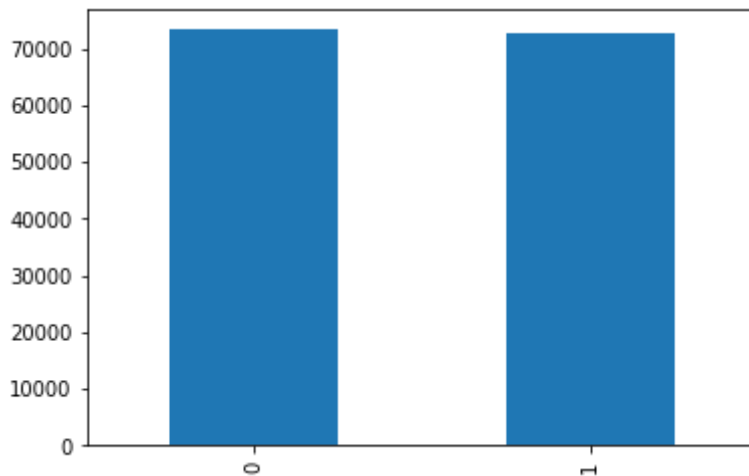
```
# document 열과 label 열의 중복을 제외한 값의 개수
train_data['document'].nunique(), train_data['label'].nunique()
# document 열의 중복 제거
train_data.drop_duplicates(subset=['document'], inplace=True)
print('총 샘플의 수 :', len(train_data))
```

총 샘플의 수 : 146183

3만마 아니었어도 별 다서 개 줘는데 왜 3만 마여서 제 신기를 분퍼하게

```
train_data['label'].value_counts().plot(kind = 'bar')
print(train_data.groupby('label').size().reset_index(name = 'count'))
```

	label	count
0	0	73342
1	1	72841



```
print(train_data.isnull().values.any())
```

True

```
print(train_data.isnull().sum())
```

```
id          0
document    1
label       0
dtype: int64
```

```
train_data.loc[train_data.document.isnull()]
```

	id	document	label
<b>25857</b>	2172111	NaN	1

```
train_data = train_data.dropna(how = 'any') # Null 값이 존재하는 행 제거
print(train_data.isnull().values.any()) # Null 값이 존재하는지 확인
```

False

```
print(len(train_data))
```

146182

```
#알파벳과 공백을 제외하고 모두 제거
```

```
eng_text = 'do!!! you expect... people~ to~ read~ the FAQ, etc. and actually accept hard~! atheism?'
print(re.sub(r'^a-zA-Z ', '', eng_text))
```

do you expect people to read the FAQ etc and actually accept hard atheism

```
# 한글과 공백을 제외하고 모두 제거
```

```
train_data['document'] = train_data['document'].str.replace("[^ㄱ-ㅎㅌ-ㅣ가-힣 ]", "")
train_data[:5]
```

/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:2: FutureWarning: The default va

	id	document	label
0	9976970	아 더빙 진짜 짜증나네요 목소리	0
1	3819312	흠포스터보고 초딩영화줄오버연기조차 가볍지 않구나	1
2	10265843	너무재밌었다그래서보는것을추천한다	0
3	9045019	교도소 이야기구면 솔직히 재미는 없다평점 조정	0
4	6483650	사이몬페그의 익살스런 연기가 돋보였던 영화스파이더맨에서 늙어보이기만	1

```
train_data['document'] = train_data['document'].str.replace('^ +', '') # white space 데이터를 empty
train_data['document'].replace('', np.nan, inplace=True)
print(train_data.isnull().sum())
```

```
id          0
document    789
label       0
dtype: int64
```

/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:1: FutureWarning: The default va  
 """Entry point for launching an IPython kernel.

```
train_data.loc[train_data.document.isnull()][:5] #null값 포함한 출력
```

	id	document	label
404	4221289	NaN	0
412	9509970	NaN	1
470	10147571	NaN	1
584	7117896	NaN	0
593	6478189	NaN	0



```
train_data = train_data.dropna(how = 'any')
print(len(train_data))
#null제거
```

145393

```
test_data.drop_duplicates(subset = ['document'], inplace=True) # document 열에서 중복인 내용이 있다
test_data['document'] = test_data['document'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣 ]", "") # 정규 표현식
test_data['document'] = test_data['document'].str.replace('^ +', "") # 공백은 empty 값으로 변경
test_data['document'].replace('', np.nan, inplace=True) # 공백은 Null 값으로 변경
test_data = test_data.dropna(how='any') # Null 값 제거
print('전처리 후 테스트용 샘플의 개수 :', len(test_data))
```

전처리 후 테스트용 샘플의 개수 : 48852

/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:2: FutureWarning: The default va

/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:3: FutureWarning: The default va  
This is separate from the ipykernel package so we can avoid doing imports until

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

```
okt = Okt()
okt.morphs('와 이런 것도 영화라고 차라리 뮤직비디오를 만드는 게 나을 뻔', stem = True)
```

['오다', '이렇다', '것', '도', '영화', '라고', '차라리', '뮤직비디오', '를', '만들다', '게',

```
X_train = []
for sentence in tqdm(train_data['document']):
    tokenized_sentence = okt.morphs(sentence, stem=True) # 토큰화
    stopwords_removed_sentence = [word for word in tokenized_sentence if not word in stopwords] # 토큰
    X_train.append(stopwords_removed_sentence)
```

76% |██████████| 110076/145393 [10:37<02:59, 196.72it/s]

```
print(X_train[:3])
```

```
X_test = []
for sentence in tqdm(test_data['document']):
    tokenized_sentence = okt.morphs(sentence, stem=True) # 토큰화
    stopwords_removed_sentence = [word for word in tokenized_sentence if not word in stopwords] # 토큰
    X_test.append(stopwords_removed_sentence)
```

... 86% |██████████| 42028/48852 [04:22<00:49, 138.23it/s]

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)
```

```
print(tokenizer.word_index)
```

```

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)

```

```

# 전체 단어 개수 중 빈도수 20이하인 단어는 제거.
# 0번 패딩 토큰을 고려하여 + 1
vocab_size = total_cnt - rare_cnt + 1
print('단어 집합의 크기 :',vocab_size)

```

```

tokenizer = Tokenizer(vocab_size)
tokenizer.fit_on_texts(X_train)
X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)

```

```
print(X_train[:3])
```

```

y_train = np.array(train_data['label'])
y_test = np.array(test_data['label'])

```

```
drop_train = [index for index, sentence in enumerate(X_train) if len(sentence) < 1]
```

```

# 빈 샘플들을 제거
X_train = np.delete(X_train, drop_train, axis=0)
y_train = np.delete(y_train, drop_train, axis=0)
print(len(X_train))
print(len(y_train))

```

```

print('리뷰의 최대 길이 :',max(len(review) for review in X_train))
print('리뷰의 평균 길이 :',sum(map(len, X_train))/len(X_train))
plt.hist([len(review) for review in X_train], bins=50)
plt.xlabel('length of samples')

```

```
plt.ylabel('number of samples')
...
def below_threshold_len(max_len, nested_list):
    count = 0
    for sentence in nested_list:
        if(len(sentence) <= max_len):
            count = count + 1
    print('전체 샘플 중 길이가 %s 이하인 샘플의 비율: %s'%(max_len, (count / len(nested_list))*100))
```

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

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

```
from tensorflow.keras.layers import Embedding, Dense, LSTM
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import load_model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

embedding_dim = 100
hidden_units = 128

model = Sequential()
model.add(Embedding(vocab_size, embedding_dim))
model.add(LSTM(hidden_units))
model.add(Dense(1, activation='sigmoid'))

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

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

```
loaded_model = load_model('best_model.h5')
print("\n 테스트 정확도: %.4f" % (loaded_model.evaluate(X_test, y_test)[1]))
```

```
def sentiment_predict(new_sentence):
    new_sentence = re.sub(r'[^ㄱ-ㅎㅌ-ㅣ가-힣 ]', '', new_sentence)
    new_sentence = okt.morphs(new_sentence, stem=True) # 토큰화
    new_sentence = [word for word in new_sentence if not word in stopwords] # 불용어 제거
    encoded = tokenizer.texts_to_sequences([new_sentence]) # 정수 인코딩
    pad_new = pad_sequences(encoded, maxlen = max_len) # 패딩
    score = float(loaded_model.predict(pad_new)) # 예측
    if(score > 0.5):
        print("{:.2f}% 확률로 긍정 리뷰입니다.\n".format(score * 100))
    else:
        print("{:.2f}% 확률로 부정 리뷰입니다.\n".format((1 - score) * 100))
```

```
sentiment_predict('이 영화 개꿀잼 ㅋㅋㅋ')
sentiment_predict('이 영화 핵노잼 ㅠㅠ')
```

```
sentiment_predict('이따게 영화냐 ㅋㅋ')
sentiment_predict('감독 뭐하는 놈이냐?')
```

#### ◆ 1-(4) 주가 예측하기

- 삼성전자 / 애플사의 주가를 예측해보겠습니다.

📌 [LSTM과 FinanceDataReader API를 활용한 삼성전자 주가 예측](#)

## ▼ 2 GRU Model

### 📖 내용 복습

- GRU Model은 LSTM Model의 장점을 유지하며 보다 간소화한 모델입니다.
- GRU Model 설명 참고 자료
  - [게이트 순환 유닛\(Gated Recurrent Unit, GRU\)](#)
  - [Gated Recurrent Units \(GRU\)](#)

😬 2-(1)~(2)의 실습 모두 실습하시면 됩니다.

#### ◆ 2-(1) IMDB 리뷰 감성 분류하기

- 해외 영화 사이트 IMDB의 리뷰 데이터를 감정에 따라 분류하고 예측해보겠습니다.

📌 [GRU로 IMDB 리뷰 감성 분류](#)

```
import re
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GRU, Embedding
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.models import load_model

vocab_size = 10000
max_len = 500

(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=vocab_size)

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

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz>  
 17465344/17464789 [=====] - 0s 0us/step  
 17473536/17464789 [=====] - 0s 0us/step

```

embedding_dim = 100
hidden_units = 128

model = Sequential()
model.add(Embedding(vocab_size, embedding_dim))
model.add(GRU(hidden_units))
model.add(Dense(1, activation='sigmoid'))

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

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

```

```

Epoch 1/15
1313/1313 [=====] - ETA: 0s - loss: 0.4778 - acc: 0.7725
Epoch 1: val_acc improved from -inf to 0.66860, saving model to GRU_model.h5
1313/1313 [=====] - 288s 910ms/step - loss: 0.4778 - acc: 0.7725 - val
Epoch 2/15
1313/1313 [=====] - ETA: 0s - loss: 0.3045 - acc: 0.8816
Epoch 2: val_acc improved from 0.66860 to 0.87780, saving model to GRU_model.h5
1313/1313 [=====] - 279s 890ms/step - loss: 0.3045 - acc: 0.8816 - val
Epoch 3/15
1313/1313 [=====] - ETA: 0s - loss: 0.2436 - acc: 0.9071
Epoch 3: val_acc improved from 0.87780 to 0.88980, saving model to GRU_model.h5
1313/1313 [=====] - 280s 895ms/step - loss: 0.2436 - acc: 0.9071 - val
Epoch 4/15
1313/1313 [=====] - ETA: 0s - loss: 0.1912 - acc: 0.9273
Epoch 4: val_acc improved from 0.88980 to 0.89400, saving model to GRU_model.h5
1313/1313 [=====] - 277s 884ms/step - loss: 0.1912 - acc: 0.9273 - val
Epoch 5/15
144/1313 [=====>.....] - ETA: 2:21 - loss: 0.1444 - acc: 0.9481

```

KeyboardInterrupt

Traceback (most recent call last)

[<ipython-input-28-e484441ff187>](#) in <module>()

```

11
12 model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
--> 13 history = model.fit(X_train, y_train, epochs=15, callbacks=[es, mc], batch_size=64,
validation_split=0.2)

```

---

 8 frames
 

---

```

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/execute.py in
quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
53     ctx.ensure_initialized()
54     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
--> 55                                     inputs, attrs, num_outputs)
56 except core._NotOkStatusException as e:
57     if name is not None:

```

KeyboardInterrupt:

```

loaded_model = load_model('GRU_model.h5')
print("\n 테스트 정확도: %.4f" % (loaded_model.evaluate(X_test, y_test)[1]))

```



```
def sentiment_predict(new_sentence):
    # 알파벳과 숫자를 제외하고 모두 제거 및 알파벳 소문자화
    new_sentence = re.sub('[^0-9a-zA-Z ]', '', new_sentence).lower()
    encoded = []

    # 띄어쓰기 단위 토큰화 후 정수 인코딩
    for word in new_sentence.split():
        try :
            # 단어 집합의 크기를 10,000으로 제한.
            if word_to_index[word] <= 10000:
                encoded.append(word_to_index[word]+3)
            else:
                # 10,000 이상의 숫자는 <unk> 토큰으로 변환.
                encoded.append(2)
        # 단어 집합에 없는 단어는 <unk> 토큰으로 변환.
        except KeyError:
            encoded.append(2)

    pad_sequence = pad_sequences([encoded], maxlen=max_len)
    score = float(loader.predict(pad_sequence)) # 예측

    if(score > 0.5):
        print("{:.2f}% 확률로 긍정 리뷰입니다.".format(score * 100))
    else:
        print("{:.2f}% 확률로 부정 리뷰입니다.".format((1 - score) * 100))
```

```
test_input = "This movie was just way too overrated. The fighting was not professional and in slow
sentiment_predict(test_input)
```

```
test_input = " I was lucky enough to be included in the group to see the advanced screening in Melb
Now, the film... how can I even begin to explain how I feel about this film? It is, as the title of
Seeing Joss Whedon's direction and envisioning of the film come to life on the big screen is perfec
sentiment_predict(test_input)
```

## ◆ 2-(2) 네이버 쇼핑 리뷰 감성 분류하기

- 네이버 쇼핑 리뷰를 감성에 따라 분류하고, 임의의 리뷰에 대해 감정을 예측해보겠습니다.

### 🔥 [GRU로 네이버 쇼핑 리뷰 감성 분류](#)

```
# Colab에 Mecab 설치
!git clone https://github.com/SOMJANG/Mecab-ko-for-Google-Colab.git
%cd Mecab-ko-for-Google-Colab
!bash install_mecab-ko_on_colab190912.sh
```

```
import re
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
import urllib.request
from collections import Counter
from konlpy.tag import Mecab
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
urllib.request.urlretrieve("https://raw.githubusercontent.com/bab2min/corpus/master/sentiment/naver",
```

```
total_data = pd.read_table('ratings_total.txt', names=['ratings', 'reviews'])
print('전체 리뷰 개수 :', len(total_data)) # 전체 리뷰 개수 출력

total_data[:5]
```

```
total_data['label'] = np.select([total_data.ratings > 3], [1], default=0)
total_data[:5]
```

```
total_data['ratings'].nunique(), total_data['reviews'].nunique(), total_data['label'].nunique()
```

```
total_data.drop_duplicates(subset=['reviews'], inplace=True) # reviews 열에서 중복인 내용이 있다면
print('총 샘플의 수 :', len(total_data))
```

```
print(total_data.isnull().values.any())
```

```
train_data, test_data = train_test_split(total_data, test_size = 0.25, random_state = 42)
print('훈련용 리뷰의 개수 :', len(train_data))
print('테스트용 리뷰의 개수 :', len(test_data))
```

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

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

```
# 한글과 공백을 제외하고 모두 제거
train_data['reviews'] = train_data['reviews'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣 ]", "")
train_data['reviews'].replace('', np.nan, inplace=True)
print(train_data.isnull().sum())
```

```
test_data.drop_duplicates(subset = ['reviews'], inplace=True) # 중복 제거
test_data['reviews'] = test_data['reviews'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣 ]", "") # 정규 표현식 수행
test_data['reviews'].replace('', np.nan, inplace=True) # 공백은 Null 값으로 변경
test_data = test_data.dropna(how='any') # Null 값 제거
print('전처리 후 테스트용 샘플의 개수 :', len(test_data))
```

```
mecab = Mecab()
print(mecab.morphs('와 이런 것도 상품이라고 차라리 내가 만드는 게 나을 뻔'))
```



```

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)

```

```

# 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)
print(X_train[:3])
print(X_test[:3])

```

```

print('리뷰의 최대 길이 :',max(len(review) for review in X_train))
print('리뷰의 평균 길이 :',sum(map(len, X_train))/len(X_train))
plt.hist([len(review) for review in X_train], bins=50)
plt.xlabel('length of samples')
plt.ylabel('number of samples')
plt.show()

```

```

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

```

```

max_len = 80
below_threshold_len(max_len, X_train)

```

```

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

```

```

from tensorflow.keras.layers import Embedding, Dense, GRU
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import load_model

```

```

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

embedding_dim = 100
hidden_units = 128

model = Sequential()
model.add(Embedding(vocab_size, embedding_dim))
model.add(GRU(hidden_units))
model.add(Dense(1, activation='sigmoid'))

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

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(X_train, y_train, epochs=15, callbacks=[es, mc], batch_size=64, validation_split=0.1)

loaded_model = load_model('best_model.h5')
print("ㄴ 테스트 정확도: %.4f" % (loaded_model.evaluate(X_test, y_test)[1]))

```

```

def sentiment_predict(new_sentence):
    new_sentence = re.sub(r'[^ㄱ-ㅎㅌ-ㅣ가-힣 ]', '', new_sentence)
    new_sentence = mecab.morphs(new_sentence)
    new_sentence = [word for word in new_sentence if not word in stopwords]
    encoded = tokenizer.texts_to_sequences([new_sentence])
    pad_new = pad_sequences(encoded, maxlen = max_len)


    score = float(loaded_model.predict(pad_new))
    if(score > 0.5):
        print("{:.2f}% 확률로 긍정 리뷰입니다.".format(score * 100))
    else:
        print("{:.2f}% 확률로 부정 리뷰입니다.".format((1 - score) * 100))

```

```

sentiment_predict('이 상품 진짜 좋아요... 저는 강추합니다. 대박')
sentiment_predict('진짜 배송도 늦고 개짜증나네요. 뭐 이런 걸 상품이라고 만듦?')

```

 17분 20초    오후 11:37에 완료됨

