11. RNN을 이용한 텍스트 분류

텍스트 분류(Text Classification)

- 분류하는 텍스트 데이터는 내용과 레이블로 구성되어 있음: 텍스트 내용을 기반으로 레이블을 분류하는 것이 목적
  - 예: 스팸 메일 분류, 뉴스 기사의 주제 분류, 영화평의 선호도 분류 등

- **이진 분류:** 두 개의 선택 중에서 정답을 고르는 형식

- **다중 클래스 분류:** 세 개 이상의 선택에서 정답을 고르는 경우

11장 내용

- Keras에서 text preprocessing 방법
- Spam mail detection
- Reuters 뉴스 분류하기
- Naïve Bayes classifier
- Naver 영화 리뷰 분류

Keras에서 입력 텍스트 처리 함수

- Tokenzer: 입력 텍스트를 숫자로 변환하는 기능을 수행

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer()
tokenizer.fit_on_texts(text_data) # 텍스트의 각 단어에 고유한 숫자를 부여
sequences = tokenizer.texts_to_sequences(text_data) # 단어를 숫자값으로 변환하여 저장
```

- 위에서 구한 sequences는 숫자 리스트로 구성되어 있음
- 전체 단어 리스트는 word_index 함수로 얻을 수 있음

```
word_to_index = tokenizer.word_index
print(word_to_index)
```

Tokenizer 적용 사례

```
from tensorflow.keras.preprocessing.text import Tokenizer
t = Tokenizer()
fit_text = "The earth is an awesome place live"
t.fit_on_texts([fit_text])

test_text = "The earth is an great place live"
sequences = t.texts_to_sequences([test_text])[0]
print("sequences : ",sequences) # great는 단어 집합에 없으므로 출력되지 않음
print("word_index : ",t.word_index) # 단어 집합(vocabulary) 출력
```

```
sequences : [1, 2, 3, 4, 6, 7]
word_index : {'the': 1, 'earth': 2, 'is': 3, 'an': 4, 'awesome': 5,
'place': 6, 'live': 7}
```

Keras에서의 텍스트 시퀀스 처리

- 신경망 입력 텍스트를 일정한 길이로 만들려면 pad_sequences 함수를 이용
- 이 함수는 sequenc의 최대 길이를 지정하여 이보다 긴 텍스트는 앞 또는 뒤를 자르고, 짧은 텍스트는 앞에 0을 채움

```
# 최대 길이를 100으로 지정하고 초과하면 각 시퀀스의 앞 쪽을 자른다
X_prime = sequence.pad_sequences(X, maxlen=100, truncating='pre')

# 최대길이를 100으로 지정하고 초과하면 각 시퀀스의 뒷 쪽을 자른다
X_prime = sequence.pad_sequences(X, maxlen=100, truncating='post')
```

pad_sequences 적용 사례

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
pad_sequences([[1, 2, 3], [3, 4, 5, 6], [7, 8]], maxlen=3, padding='pre')
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [0, 7, 8]], dtype=int32)
```

워드 임베딩

- 입력 텍스트를 밀집 벡터로 표현하는 수단으로 Keras에서는 Embedding 레이어를 제공
- Embedding 층은 정수 인코딩이 된 단어들을 임베딩 벡터로 변환
- Embedding 호출시 단어수와 임베딩 벡터 길이를 입력으로 전달
- Embedding 출력은 (단어수, 벡터 길이, 입력 시퀀스 길이)의 3D 텐서를 리턴
- Word2Vec과 같은 실제 임베딩 벡터를 사용할 수도 있지만, 여기서는 자체적으로 생성하는 벡터임

워드 임베딩 사례

```
# 문장 토큰화와 단어 토큰화
text=[['Hope', 'to', 'see', 'you', 'soon'],['Nice', 'to', 'see', 'you', 'again']]
# 각 단어에 대한 정수 인코딩
text=[[0, 1, 2, 3, 4],[5, 1, 2, 3, 6]]
# 위 데이터가 아래의 임베딩 층의 입력이 된다.
Embedding(7, 2, input_length=5)
# 7은 단어의 개수. 즉, 단어 집합(vocabulary)의 크기이다.
# 2는 임베딩한 후의 벡터의 크기이다.
# 5는 각 입력 시퀀스의 길이. 즉, input_length이다.
# 각 정수는 아래의 테이블의 인덱스로 사용되며 Embeddig()은 각 단어에 대해 임베딩 벡터를 리턴
```

index	embedding
0	[1.2, 3.1]
1	[0.1, 4.2]
2	[1.0, 3.1]
3	[0.3, 2.1]
4	[2.2, 1.4]
5	[0.7, 1.7]
6	[4.1, 2.0]

스팸 메일 분류하기(Spam Detection)

- 메일의 내용을 이용하여 스팸 메일 여부를 판단
- 이 예제에서는 데이터 전처리를 진행하고 RNN을 이용하여 스팸 메일을 분류

텍스트(메일의 내용)	레이블(스팸 여부)
당신에게 드리는 마지막 혜택! ...	스팸 메일
내일 뵐 수 있을지 확인 부탁...	정상 메일
쉿! 혼자 보세요...	스팸 메일
언제까지 답장 가능할...	정상 메일
...	...
(광고) 멋있어질 수 있는...	스팸 메일

스팸 메일 데이터

- 메일 데이터는 kaggle 사이트에서 읽음

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

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')
```

데이터 보기

- 총 5,572개의 메일을 포함하고 있음
- 'v1' 열에 spam 여부가 나와 있고, 'v2'에 메일 내용이 있음: 'v2'를 이용하여 'v1'을 유도하는 것이 이 예제의 목적임

```
print('총 샘플의 수 :',len(data))

print(data[:5])
```

```
총 샘플의 수 : 5572
   v1                  v2 Unnamed: 2                \
0  ham  Go until jurong point, crazy.. Available only ...  NaN
1  ham  Ok lar... Joking wif u oni...                     NaN
2  spam  Free entry in 2 a wkly comp to win FA Cup fina...  NaN
3  ham  U dun say so early hor... U c already then say...  NaN
4  ham  Nah I don't think he goes to usf, he lives aro...  NaN
```

스팸 여부

메일 내용

메일 텍스트를 숫자로 변환

- Tokenizer를 이용하여 메일 내용을 숫자로 바꿈
- 분류에 사용할 수 있게 'v1' 텍스트를 [0, 1] 숫자로 변환
- 메일 내용의 길이가 일정하도록 pad_sequences를 수행

메일 텍스트 변환 부분

```
data['v1'] = data['v1'].replace(['ham','spam'],[0,1])   # 메일 유형을 [0, 1]로 변환

X_data = data['v2']          # 입력 메일 내용
y_data = data['v1']          # 메일 유형

tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_data)       # X의 각 행에 토큰화를 수행
sequences = tokenizer.texts_to_sequences(X_data) # 단어를 숫자값으로 변환하여 저장

word_to_index = tokenizer.word_index # 사용된 단어 리스트를 보관
vocab_size = len(word_to_index) + 1
n_of_train = int(len(sequences) * 0.8)         # 데이터의 80%를 훈련용으로 이용
n_of_test = int(len(sequences) - n_of_train)    # 데이터의 20%를 시험용으로 이용

X_data = sequences

max_len = max(len(l) for l in X_data)          # 가장 긴 메일의 길이를 max_len에 저장
data = pad_sequences(X_data, maxlen = max_len) # 전체 데이터셋의 길이는 max_len으로

X_test = data[n_of_train:]  #X_data 데이터 중에서 뒤의 20%의 데이터만 저장
y_test = np.array(y_data[n_of_train:]) #y_data 데이터 중에서 뒤의 20%의 데이터만
X_train = data[:n_of_train]          #X_data 데이터 중에서 앞의 4135개의 데이터만
y_train = np.array(y_data[:n_of_train]) #y_data 데이터 중에서 앞의 4135개의 데이터
```

신경망 생성

- 입력단으로 Embedding 레이어를 사용
- RNN 구조를 다음 단으로 사용
- 출력단은 1개의 sigmoid 단으로 구성

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

model = Sequential()
model.add(Embedding(vocab_size, 32)) # 임베딩 벡터의 차원은 32
model.add(SimpleRNN(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, y_train, epochs=4, batch_size=64,
        validation_split=0.2)
```

신경망 구조

- model.summary() 함수를 이용하여 구조를 볼 수 있음

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 32)          285472
_________________________________________________________________
simple_rnn_2 (SimpleRNN)     (None, 32)                2080
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 287,585
Trainable params: 287,585
Non-trainable params: 0
```

- Embedding layer에서는 vocab_size x 32 개의 파라미터가 있음

전체 프로그램: 1

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import urllib.request
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

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')

print('총 샘플의 수 :',len(data))

print(data[:5])
data['v1'] = data['v1'].replace(['ham','spam'],[0,1])

X_data = data['v2']
y_data = data['v1']

tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_data) # X의 각 행에 토큰화를 수행
sequences = tokenizer.texts_to_sequences(X_data) # 단어를 숫자값, 인덱스로 변환하여 저장

word_to_index = tokenizer.word_index

vocab_size = len(word_to_index) + 1
```

전체 프로그램: 2

```
n_of_train = int(len(sequences) * 0.8)
n_of_test = int(len(sequences) - n_of_train)

X_data = sequences

max_len = max(len(l) for l in X_data)
# 전체 데이터셋의 길이는 max_len으로 맞춥니다.
data = pad_sequences(X_data, maxlen = max_len)
print("훈련 데이터의 크기(shape): ", data.shape)

X_test = data[n_of_train:] #X_data 데이터 중에서 뒤의 1034개의 데이터만 저장
y_test = np.array(y_data[n_of_train:]) #y_data 데이터 중에서 뒤의 1034개의 데이터만 저장
X_train = data[:n_of_train] #X_data 데이터 중에서 앞의 4135개의 데이터만 저장
y_train = np.array(y_data[:n_of_train]) #y_data 데이터 중에서 앞의 4135개의 데이터만 저장

from tensorflow.keras.layers import SimpleRNN, Embedding, Dense
from tensorflow.keras.models import Sequential

model = Sequential()
model.add(Embedding(vocab_size, 32)) # 임베딩 벡터의 차원은 32
model.add(SimpleRNN(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, y_train, epochs=4, batch_size=64, validation_split=0.2)
```

Reuters 뉴스 분류하기

- Keras에서 제공하는 로이터 뉴스 데이터를 LSTM을 이용하여 분류
- 총11,258 개의 기사가 46개의 뉴스 카테고리로 분류되어 있음

```
from tensorflow.keras.datasets import reuters
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

(X_train, y_train), (X_test, y_test) =
        reuters.load_data(num_words=None, test_split=0.2)
```

- 데이터를 읽을 때 num_words 파라미터로 사용할 단어의 숫자를 입력함
- 읽은 기사 X_train은 숫자 시퀀스로 들어오므로 내용을 보려면 별도의 프로그램이 필요

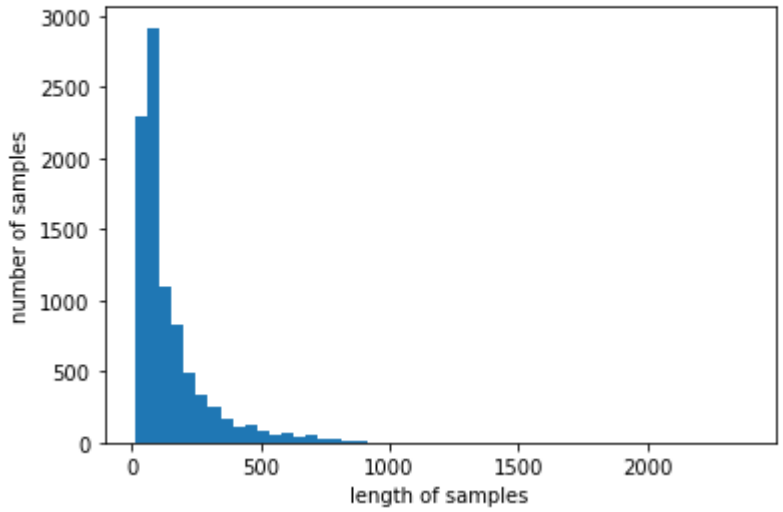
뉴스 데이터

- 뉴스 데이터로부터 기사들의 범주를 결정하는 것이 이 예제의 목적임
- 다음은 기사의 길이를 조사하는 프로그램

```
print('뉴스 기사의 최대 길이 :{}'.format(max(len(l) for l in X_train)))
print('뉴스 기사의 평균 길이 :{}'.format(sum(map(len, X_train))/len(X_train)))

plt.hist([len(s) for s in X_train], bins=50)
plt.xlabel('length of samples')
plt.ylabel('number of samples')
plt.show()
```

```
뉴스 기사의 최대 길이 :2376
뉴스 기사의 평균 길이 :145.5398574927633
```



뉴스 기사 보기

- get_word_index 함수를 이용하여 숫자와 단어의 관계를 구할 수 있음
- 0번 기사의 내용은 다음과 같이 볼 수 있음

```
word_to_index = reuters.get_word_index()
#print(word_to_index)
index_to_word = {}
for key, value in word_to_index.items():
    index_to_word[value] = key

for index, token in enumerate(("<pad>", "<sos>", "<unk>")):
  index_to_word[index]=token

print(' '.join([index_to_word[index] for index in X_train[0]]))
```

```
<sos> wattie nondiscriminatory mln loss for plc said at only ended said
commonwealth could 1 traders now april 0 a after said from 1985 and from
foreign 000 april 0 prices its account year a but in this mln home an
states earlier and rise and revs vs 000 its 16 vs 000 a but 3 psbr oils
several and shareholders and dividend vs 000 its all 4 vs 000 1 mln agreed
largely april 0 are 2 states will billion total and against 000 pct dlrs
```

LSTM으로 뉴스 분류하기

- 여기서는 뉴스 데이터를 읽을 때 1000개의 단어만을 사용

```
from tensorflow.keras.datasets import reuters
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Embedding
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.models import load_model

(X_train, y_train), (X_test, y_test) = reuters.load_data(num_words=1000,
test_split=0.2)
```

데이터 정리

- 각 기사의 길이는 100 단어로 하여 sequence로 만듦
- 기사 유형은 one-hot vector로 표현

```
max_len = 100
X_train = pad_sequences(X_train, maxlen=max_len) # 훈련용 뉴스 기사 패딩
X_test = pad_sequences(X_test, maxlen=max_len) # 테스트용 뉴스 기사 패딩

y_train = to_categorical(y_train) # 훈련용 뉴스 기사 레이블의 원-핫 인코딩
y_test = to_categorical(y_test) # 테스트용 뉴스 기사 레이블의 원-핫 인코딩
```

신경망 구조

- 첫 단은 Embedding 레이어로 구현
- 다음 단은 120셀의 LSTM으로 구현
- 출력단은 46개의 셀로 구성

```
model = Sequential()
model.add(Embedding(1000, 120))
model.add(LSTM(120))
model.add(Dense(46, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam',
        metrics=['acc'])
history = model.fit(X_train, y_train, batch_size=128, epochs=20,
        validation_data=(X_test, y_test))
```

신경망 구조

- model.summary 출력

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 120)         120000
_________________________________________________________________
lstm (LSTM)                  (None, 120)               115680
_________________________________________________________________
dense (Dense)                (None, 46)                5566
=================================================================
Total params: 241,246
Trainable params: 241,246
Non-trainable params: 0
```

전체 프로그램

```
from tensorflow.keras.datasets import reuters
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Embedding
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import load_model

(X_train, y_train), (X_test, y_test) = reuters.load_data(num_words=1000,
test_split=0.2)

max_len = 100
X_train = pad_sequences(X_train, maxlen=max_len) # 훈련용 뉴스 기사 패딩
X_test = pad_sequences(X_test, maxlen=max_len) # 테스트용 뉴스 기사 패딩

y_train = to_categorical(y_train) # 훈련용 뉴스 기사 레이블의 원-핫 인코딩
y_test = to_categorical(y_test) # 테스트용 뉴스 기사 레이블의 원-핫 인코딩

model = Sequential()
model.add(Embedding(1000, 120))
model.add(LSTM(120))
model.add(Dense(46, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])

history = model.fit(X_train, y_train, batch_size=128, epochs=20,
validation_data=(X_test, y_test))
```

Naïve Bayes Classifier

- 확률에서의 베이즈 원리를 이용하여 텍스트 내용을 이용하여 클래스를 분류하는 것
- 교재의 11.5절에서는 나이브 베이즈 분류를 이용하여 스팸 메일 여부를 판단하는 프로그램을 구성
- 분류기는 sklearn 패키지에서 제공되는 함수를 이용함

Naïve Bayes Model

- 문서 d가 주어졌을 때 이 것을 범주 c_1 또는 c_2로 구분하고자 함
- Bayes 법칙

$$P(c_1|d) = \frac{P(c_1,d)}{P(d)} = \frac{P(d|c_1)P(c_1)}{P(d)}$$

$$P(c_2|d) = \frac{P(c_2,d)}{P(d)} = \frac{P(d|c_2)P(c_2)}{P(d)}$$

$$P(B|A) = \frac{P(A|B)P(B)}{P(B \cap A)} \quad P(A)$$

- 문서 d로 부터 위 두 확률을 구하여 확률이 큰 범주를 선택하는 것이 Bayes 방식임
- 예를 들면 메일 d의 스팸 여부를 결정하려면 $P(spam|d)$ 과 $P(ham|d)$을 계산하여 확률이 큰 쪽을 선택하는 것임

조건부 확률 계산

- 앞의 확률식에서 분모 $P(d)$는 공통이므로 분자만 고려해도 됨

$$P(c_i \mid d) \propto P(d \mid c_i)P(c_i)$$

- 문서 d가 단어 w_1, w_2로 구성되어 있다고 가정하면

$$P(c_i \mid d) \propto P(w_1, w_2 \mid c_i)P(c_i)$$

$$= P(w_1 \mid c_i)P(w_2 \mid c_i)P(c_i)$$

- Naïve Bayes 모델에서는 각 단어의 발생 확률이 서로 독립이라고 가정해서 위의 관계식이 얻어진 것임

Bag of words 단어 표현

- Naïve Bayes 방식에서는 BOW에서 각 단어의 발생 확률만을 고려하는 것과 유사하게 문서에서 각 단어의 발생 확률을 이용하여 조건부 확률을 구함

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1

계산 사례

- 다음과 같은 영화평이 있다고 가정. – 는 부정적, + 는 긍정적 평가를 의미

	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable with no fun

- 평가의 발생 확률: $P(-) = 3/5, P(+) = 2/5$
- 부정 범주 단어 발생 확률

$$P(predictable|-) = \frac{1}{14}$$

$$P(no|-) = \frac{1}{14}$$

$$P(fun|-) = \frac{0}{14}$$

긍정 범주 단어 확률

$$P(predictable|+) = \frac{0}{9}$$

$$P(no|+) = \frac{0}{9}$$

$$P(fun|+) = \frac{1}{9}$$

예측 결과

- 두 범주에 대해 아래 확률을 계산하여 큰 쪽을 선택

$$P(-) \times P(predictable|-) \times P(no|-) \times P(fun|-)$$
$$P(+) \times P(predictable|+) \times P(no|+) \times P(fun|+)$$

- 이 예제의 경우 훈련시 나타나지 않아 확률이 0인 단어가 있어서 이 방식을 사용하지 못함
- 훈련시 나타나지 않은 단어는 예측할 때 제외하는 방식을 많이 사용함

뉴스 그룹 데이터 분류하기

- 여기서는 사이킷 런(sklearn)에서 제공하는 20개의 다른 주제를 가진 뉴스 데이터를 나이브 베이즈 기법으로 분류하는 예제를 분석
- 훈련 데이터는 다음과 같이 읽음

```
from sklearn.datasets import fetch_20newsgroups
newsdata=fetch_20newsgroups(subset='train')
print(newsdata.keys())
```

- newsdata.data에는 텍스트, newsdata.target_names에는 주제가 들어 있음

텍스트 처리

- sklearn에서 제공하는 MultinomialNB 함수를 이용
- 텍스트 데이터는 tf-idf 형식으로 변환하여 입력으로 사용
- 이 예제에서는 데이터 전처리를 진행하고 RNN을 이용하여 스팸 메일을 분류

전체 프로그램

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB # 다항분포 나이브 베이즈 모델
from sklearn.metrics import accuracy_score #정확도 계산

dtmvector = CountVectorizer()
X_train_dtm = dtmvector.fit_transform(newsdata.data)

tfidf_transformer = TfidfTransformer()
tfidfv = tfidf_transformer.fit_transform(X_train_dtm) # 입력을 tf-idf로 변환
print(tfidfv.shape)

mod = MultinomialNB()
mod.fit(tfidfv, newsdata.target)

MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)

newsdata_test = fetch_20newsgroups(subset='test', shuffle=True) #테스트 데이터 갖고오기
X_test_dtm = dtmvector.transform(newsdata_test.data) #테스트 데이터를 DTM으로 변환
tfidfv_test = tfidf_transformer.transform(X_test_dtm) #DTM을 TF-IDF 행렬로 변환

predicted = mod.predict(tfidfv_test) #테스트 데이터에 대한 예측
print("정확도:", accuracy_score(newsdata_test.target, predicted)) #예측값과 실제값 비교
```

네이버 영화 리뷰 분석

- 네이버에서 준비한 영화 리뷰를 분석하여 긍정/부정 평가 범주를 구분함
- 한글로 된 영화평을 정리하기 위해 데이터에 대한 사전 정리를 수행
- 데이터 읽어 오기: 150,000개의 데이터가 table 형식으로 되어 있음

```
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')

train_data[:5]
```

	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

분석 방식

- ‘document’ 필드에 있는 영화평으로부터 ‘label’에 있는 평가(0 또는 1)를 예측함
- 영화평에서 중복 데이터를 제거하고, 한글 데이터 이외의 문자들을 삭제함

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
import urllib.request
from konlpy.tag import Okt
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",
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')
```

텍스트 전처리

```
train_data.drop_duplicates(subset=['document'], inplace=True) # document 열에서 중복인 내용이 있다면 중복 제거

print('총 샘플의 수 :',len(train_data))
print(train_data.groupby('label').size().reset_index(name = 'count'))

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

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

train_data = train_data.dropna(how = 'any')
print(len(train_data))

test_data.drop_duplicates(subset = ['document'], inplace=True) # document 중복 제거
test_data['document'] = test_data['document'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣 ]","")
# 정규 표현식 수행
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))
```

한글 형태소 분석

- X_train(훈련)과 X_test(시험) 데이터를 토큰화하고 불용어를 제거
- 프로그램 수행 시간이 4~5분 걸림

```
stopwords =
['의','가','이','은','들','는','좀','잘','걍','과','도','를','으로','자','에','와'
,'한','하다']
okt = Okt()       # 형태소 분석기
X_train = []
k = 0
for sentence in train_data['document']:
    k = k+1
    if k % 5000 == 0: print(k)
    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)
X_test = []
k = 0
for sentence in test_data['document']:
    k = k+1
    if k % 5000 == 0: print(k)
    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_test.append(temp_X)
```

문장을 30 단어 길이의 시퀀스화

- model.summary() 함수를 이용하여 구조를 볼 수 있음

```
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)

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]

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))

max_len = 30
X_train = pad_sequences(X_train, maxlen = max_len)
X_test = pad_sequences(X_test, maxlen = max_len)
```

신경망 구조

- 입력 문장을 100자리의 Embedding 레이어로 변환
- 100 개의 셀로 구성된 LSTM으로 연결
- 출력단은 1개의 셀로 구성

```
model = Sequential()
model.add(Embedding(vocab_size, 100))
model.add(LSTM(128))
model.add(Dense(1, activation='sigmoid'))
```

리뷰 예측

- 새로 작성한 영화평에 대해 프로그램에서 수행한 전처리를 적용하고 리뷰에 대한 점수를 산출

```
def sentiment_predict(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')
97.76% 확률로 긍정 리뷰입니다.

sentiment_predict('이 영화 핵노잼 ㅠㅠ')
98.55% 확률로 부정 리뷰입니다.

sentiment_predict('이딴게 영화냐 ㅉㅉ')
99.91% 확률로 부정 리뷰입니다.
```