Chapter4. 텍스트 분류

Ⅰ. 영어 텍스트 분류

https://www.kaggle.com/c/word2vec-nlp-tutorial/

위 페이지에서 'Bag of Words Meets Bag of Popcorn' 데이터를 이용하였다. 영화 평점 데이터이다.

1. 데이터 정보 확인

```
print('리뷰 길이 최댓값: {}'.format(np.max(train_length)))
print('리뷰 길이 최솟값: {}'.format(np.min(train_length)))
print('리뷰 길이 평균값: {:.2f}'.format(np.mean(train_length)))
print('리뷰 길이 표준편차: {:.2f}'.format(np.std(train_length)))
print('리뷰 길이 중간값: {}'.format(np.median(train_length)))
print('리뷰 길이 제1사분위: {}'.format(np.percentile(train_length, 25)))
print('리뷰 길이 제3사분위: {}'.format(np.percentile(train_length, 75)))
```

```
리뷰 길이 최댓값: 13710
리뷰 길이 최솟값: 54
리뷰 길이 평균값: 1329.71
리뷰 길이 표준편차: 1005.22
리뷰 길이 중간값: 983.0
리뷰 길이 제1사분위: 705.0
리뷰 길이 제3사분위: 1619.0
```

```
print('리뷰 단어 개수 최댓값: {}'.format(np.max(train_word_counts)))
print('리뷰 단어 개수 최솟값: {}'.format(np.min(train_word_counts)))
print('리뷰 단어 개수 평균값: {:.2f}'.format(np.mean(train_word_counts)))
print('리뷰 단어 개수 표준편차: {:.2f}'.format(np.std(train_word_counts)))
print('리뷰 단어 개수 중간값: {}'.format(np.median(train_word_counts)))
print('리뷰 단어 개수 제1사분위: {}'.format(np.percentile(train_word_counts, 25)))
print('리뷰 단어 개수 제3사분위: {}'.format(np.percentile(train_word_counts, 75)))
```

```
리뷰 단어 개수 최댓값: 2470
리뷰 단어 개수 최솟값: 10
리뷰 단어 개수 평균값: 233.79
리뷰 단어 개수 표준편차: 173.74
리뷰 단어 개수 중간값: 174.0
리뷰 단어 개수 제1사분위: 127.0
리뷰 단어 개수 제3사분위: 284.0
```

```
qmarks=np.mean(train_data['review'].apply(lambda x:'?' in x))
fullstop=np.mean(train_data['review'].apply(lambda x:'.' in x))
capital_first=np.mean(train_data['review'].apply(lambda x:x[0].isupper()))
capitals=np.mean(train_data['review'].apply(lambda x:max([y.isupper() for y in x])))
numbers=np.mean(train_data['review'].apply(lambda x:max([y.isdigit() for y in x])))
print('물음표가 있는 질문: {:.2f}%'.format(qmarks*100))
print('마침표가 있는 질문: {:.2f}%'.format(fullstop*100))
print('첫 글자가 대문자인 질문: {:.2f}%'.format(capital_first*100))
print('대문자가 있는 질문: {:.2f}%'.format(capitals*100))
print('숫자가 있는 질문: {:.2f}%'.format(numbers*100))
```

```
물음표가 있는 질문: 29.55%
마침표가 있는 질문: 99.69%
첫 글자가 대문자인 질문: 0.00%
대문자가 있는 질문: 99.59%
숫자가 있는 질문: 56.66%
```

2. 데이터 전처리

불용어 제거, html 태그 제거, 대문자를 소문자로 바꾸기

```
def preprocessing(review, remove_stopwords=False):
  review_text=BeautifulSoup(review, "html5lib").get_text()
  review_text=re.sub("^[a-zA-Z]", " ", review_text)
  words=review_text.lower().split()
  if remove_stopwords:
    stops=set(stopwords.words("english"))
    words=[w for w in words if not w in stops]
    clean_review=' '.join(words)
  else:
    clean_review=' '.join(words)
  return clean_review
```

```
clean_train_reviews=[]
for review in train_data['review']:
  clean_train_reviews.append(preprocessing(review, remove_stopwords=True))
clean_train_reviews[0]
```

리뷰를 텍스트가 아닌 인덱스의 벡터로 구성

```
tokenizer=Tokenizer()
tokenizer.fit_on_texts(clean_train_reviews)
text_sequences=tokenizer.texts_to_sequences(clean_train_reviews)
```

단어 사전 확인, 단어 사전과 전체 단어 개수 딕셔너리에 저장

```
word_vocab=tokenizer.word_index
word_vocab["<PAD>"]=0
print(word_vocab)
```

```
{'movie': 1, 'film': 2, 'one': 3, 'like': 4, 'good': 5, 'the': 6, 'it': 7, 'time': 8, 'e
```

```
data_configs={}
data_configs['vocab']=word_vocab
data_configs['vocab_size']=len(word_vocab)
```

```
MAX_SEQUENCE_LENGTH=174
train_inputs=pad_sequences(text_sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post')
print('Shape of train data: ', train_inputs.shape)
```

```
Shape of train data:  (25000, 174)
```

```
train_labels=np.array(train_data['sentiment'])
print('Shape of label tensor: ', train_labels.shape)
```

```
Shape of label tensor:  (25000,)
```

전처리 한 것과 단어 사전을 파일로 저장

```
DATA_IN_PATH='/content/drive/MyDrive/Kaggle/word2vec-nlp-tutorial/'
TRAIN_INPUT_DATA='train_input.npy'
TRAIN_LABEL_DATA='train_label.npy'
TRAIN_CLEAN_DATA='train_clean.csv'
DATA_CONFIGS='data_configs.json'

import os
if not os.path.exists(DATA_IN_PATH):
  os,makedirs(DATA_IN_PATH)
```

```
np.save(open(DATA_IN_PATH+TRAIN_INPUT_DATA, 'wb'), train_inputs)
np.save(open(DATA_IN_PATH+TRAIN_LABEL_DATA, 'wb'), train_labels)
clean_train_df.to_csv(DATA_IN_PATH+TRAIN_CLEAN_DATA, index=False)
json.dump(data_configs, open(DATA_IN_PATH+DATA_CONFIGS,'w'), ensure_ascii=False)
```

Test data에 대해서도 위와 같이 똑같이 해준다.

3. 로지스틱 회귀모델 1) TF_-IDF

TF-IDF 벡터화

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer=TfidfVectorizer(min_df=0.0, analyzer="char", sublinear_tf=True, ngram_range=(1, 3), max_features=5000)
X=vectorizer.fit_transform(reviews)
```

학습과 검증 데이터셋 분리

```
from sklearn.model_selection import train_test_split
import numpy as np
RANDOM_SEED=42
TEST_SPLIT=0.2
y=np.array(sentiments)

X_train, X_eval, y_train, y_eval=train_test_split(X, y, test_size=TEST_SPLIT, random_state=RANDOM_SEED)
```

모델 선언 및 학습

```
from sklearn.linear_model import LogisticRegression
lgs=LogisticRegression(class_weight='balanced')
lgs.fit(X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

검증 데이터로 성능 평가

```
print("Accuracy: %f"%lgs.score(X_eval, y_eval))
```

```
Accuracy: 0.863000
```

결과 데이터를 kaggle에 제출한 결과 점수가 아래와 같이 나왔다.

4. 로지스틱 회귀모델 2)word2vec

Word2vec 벡터화

```
num_features=300 #워드 벡터 특징값 수, 각 단어에 대해 임베딩된 벡터의 차원
min_word_count=40 #단어에 대한 최소 빈도 수의 단어들은 학습하지 않는다
num_workers=4 #프로세스 개수
context=10 #context window 크기
downsampling=1e-3 #down sampling 비율
```

```
!pip install gensim
```

```
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s',level=logging.INFO)
```

```
from gensim.models import word2vec
print("Training model...")
model=word2vec.Word2Vec(sentences, workers=num_workers, size=num_features, min_count=min_word_count, window=context, sample=downsampling)
```

```
model_name="300features_40minwords_10context"
model.save(model_name)
```

```
def get_features(words, model, num_features):
  #word : 단어의 모음인 하나의 리뷰
  #model : word2vec 모델
  #num_features : word2vec으로 임베딩 할 때 정했던 벡터의 차원의 수
  feature_vector=np.zeros((num_features), dtype=np.float32)
  num_words=0
  index2word_set=set(model.wv.index2word)
  for w in words:
    if w in index2word_set:
      num_words+=1
      feature_vector=np.add(feature_vector, model[w])
  feature_vector=np.divide(feature_vector, num_words)
  return feature_vector
```

```
def get_dataset(reviews, model, num_features):
  #reviews : 학습 데이터 전체인 리뷰 데이터
  #model : word2vec 모델
  #num_features : word2vec으로 임베딩 할 때 정했떤 벡터의 차원 수
  dataset=list()
  for s in reviews:
    dataset.append(get_features(s, model, num_features))
  reviewFeatureVecs=np.stack(dataset)
  return reviewFeatureVecs
```

```
test_data_vecs=get_dataset(sentences, model, num_features)
```

학습과 검증 데이터셋 분리

```
from sklearn.model_selection import train_test_split
import numpy as np

X=test_data_vecs
y=np.array(sentiments)

RANDOM_SEED=42
TEST_SPLIT=0.2

X_train, X_eval, y_train, y_eval=train_test_split(X, y, test_size=TEST_SPLIT, random_state=RANDOM_SEED)
```

모델 선언 및 학습

```
from sklearn.linear_model import LogisticRegression
lgs2=LogisticRegression(class_weight='balanced')
lgs2.fit(X_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
LogisticRegression(C=1.0, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

검증 데이터셋을 이용한 성능 평가

```
print("Accuracy: %f"%lgs2.score(X_eval, y_eval))
```

```
Accuracy: 0.837800
```

결과 데이터를 kaggle에 제출한 결과 점수가 아래와 같이 나왔다.

5. 랜덤포레스트 분류 모델

CountVectorizer를 활용한 벡터화

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer=CountVectorizer(analyzer='word', max_features=5000)
train_data_features=vectorizer.fit_transform(reviews)
```

학습과 검증 데이터 분리

```
from sklearn.model_selection import train_test_split
TEST_SIZE=0.2
RANDOM_SEED=42
train_input, eval_input, train_label, eval_label=train_test_split(train_data_features, y, test_size=TEST_SIZE, random_state=RANDOM_SEED)
```

모델 구현 및 학습

```
from sklearn.ensemble import RandomForestClassifier

forest=RandomForestClassifier(n_estimators=100)
forest.fit(train_input, train_label)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

```
print("Accuracy: %f"%forest.score(eval_input, eval_label))
```

```
Accuracy: 0.843400
```

결과 데이터를 kaggle에 제출한 결과 점수가 아래와 같이 나왔다.

Name	Submitted	Wait time	Execution time	Score
randomforest.csv	just now	2 seconds	0 seconds	0.84228

Complete

6. 순환 신경망 분류 모델

랜덤 씨드 고정

```
import tensorflow as tf
SEED_NUM=1234
tf.random.set_seed(SEED_NUM)
```

모델 하이퍼파라미터 정의

```
model_name='rnn_classifier_en'
BATCH_SIZE=128
NUM_EPOCHS=5
VALID_SPLIT=0.1
MAX_LEN=train_input.shape[1]

kargs={'model_name':model_name, 'vocab_size':prepro_configs['vocab_size'], 'embedding_dimension':100,
       'dropout_rate':0.2, 'lstm_dimension':150, 'dense_dimension':150, 'output_dimension':1}
```

모델 구현

```
class RNNClassifier(tf.keras.Model):
  def __init__(self, **kargs):
    super(RNNClassifier, self).__init__(name=kargs['model_name'])
    self.embedding=tf.keras.layers.Embedding(input_dim=kargs['vocab_size'], output_dim=kargs['embedding_dimension'])
    self.lstm_1_layer=tf.keras.layers.LSTM(kargs['lstm_dimension'],return_sequences=True)
    self.lstm_2_layer=tf.keras.layers.LSTM(kargs['lstm_dimension'])
    self.dropout=tf.keras.layers.Dropout(kargs['dropout_rate'])
    self.fc1=tf.keras.layers.Dense(units=kargs['dense_dimension'], activation=tf.keras.activations.tanh)
    self.fc2=tf.keras.layers.Dense(units=kargs['output_dimension'], activation=tf.keras.activations.sigmoid)

  def call(self, x):
    x=self.embedding(x)
    x=self.dropout(x)
    x=self.lstm_1_layer(x)
    x=self.lstm_2_layer(x)
    x=self.dropout(x)
    x=self.fc1(x)
    x=self.dropout(x)
    x=self.fc2(x)
    return x
```

모델 생성

```
model=RNNClassifier(**kargs)
model.compile(optimizer=tf.keras.optimizers.Adam(1e-4), loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy')])
```

모델 학습

```
import os
earlystop_callback=tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', min_delta=0.0001, patience=2)
checkpoint_path=DATA_IN_PATH+model_name+'/weight.h5'
checkpoint_dir=os.path.dirname(checkpoint_path)
```

```
cp_callback=tf.keras.callbacks.ModelCheckpoint(checkpoint_path, monitor='val_accuracy',
                                               verbose=1, save_best_only=True, save_weights_only=True)
history=model.fit(train_input, train_label, batch_size=BATCH_SIZE, epochs=NUM_EPOCHS,
                  validation_split=VALID_SPLIT, callbacks=[earlystop_callback, cp_callback])
```

```
Epoch 1/5
176/176 [==============================] - 310s 2s/step - loss: 0.0512 - accuracy: 0.9860 - val_loss: 0.4264 - val_accuracy: 0.8736

Epoch 00001: val_accuracy improved from -inf to 0.87360, saving model to /content/drive/MyDrive/Kaggle/word2vec-nlp-tutorial/rnn_clas
Epoch 2/5
176/176 [==============================] - 311s 2s/step - loss: 0.0327 - accuracy: 0.9920 - val_loss: 0.4313 - val_accuracy: 0.8812

Epoch 00002: val_accuracy improved from 0.87360 to 0.88120, saving model to /content/drive/MyDrive/Kaggle/word2vec-nlp-tutorial/rnn_c
Epoch 3/5
176/176 [==============================] - 310s 2s/step - loss: 0.0183 - accuracy: 0.9960 - val_loss: 0.6289 - val_accuracy: 0.8756

Epoch 00003: val_accuracy did not improve from 0.88120
Epoch 4/5
176/176 [==============================] - 309s 2s/step - loss: 0.0172 - accuracy: 0.9962 - val_loss: 0.5895 - val_accuracy: 0.8708

Epoch 00004: val_accuracy did not improve from 0.88120
```

데이터 제출한 결과 아래와 같다.

```
TEST_INPUT_DATA='test_input.npy'
TEST_ID_DATA='test_id.npy'
SAVE_FILE_NM='weight.h5'

test_input=np.load(open(DATA_IN_PATH+TEST_INPUT_DATA, 'rb'))
test_input=tf.keras.preprocessing.sequence.pad_sequences(test_input, maxlen=test_input.shape[1])
model.load_weights(os.path.join(DATA_IN_PATH, model_name, SAVE_FILE_NM))
```

```
test_input=np.load(open(DATA_IN_PATH+TEST_INPUT_DATA, 'rb'))
```

```
import pandas as pd
predictions=model.predict(test_input, batch_size=BATCH_SIZE)
predictions=predictions.squeeze(-1)

test_id=np.load(open(DATA_IN_PATH+TEST_ID_DATA, 'rb'), allow_pickle=True)
if not os.path.exists(DATA_IN_PATH):
  os.makedirs(DATA_IN_PATH)
output=pd.DataFrame(data={"id":list(test_id), "sentiment":list(predictions)})
output.to_csv(DATA_IN_PATH+"movie_review_result_rnn.csv", index=False, quoting=3)
```

Name	Submitted	Wait time	Execution time	Score
movie_review_result_rnn.csv	just now	1 seconds	1 seconds	0.51596

Complete

7. 합성곱 신경망 분류 모델

모델 구현

```
model_name='cnn_classifier_en'
BATCH_SIZE=512
NUM_EPOCHS=2
VALID_SPLIT=0.1
MAX_LEN=train_input.shape[1]

kargs={'model_name':model_name, 'vocab_size':prepro_configs['vocab_size'], 'embedding_size':128,
       'num_filters':100, 'dropout_rate':0.5, 'hidden_dimension':250, 'output_dimension':1}
```

```
class CNNClassifier(tf.keras.Model):
  def __init__(self, **kargs):
    super(CNNClassifier, self).__init__(name=kargs['model_name'])
    self.embedding=tf.keras.layers.Embedding(input_dim=kargs['vocab_size'], output_dim=kargs['embedding_size'])
    self.conv_list=[tf.keras.layers.Conv1D(filters=kargs['num_filters'], kernel_size=kernel_size, padding='valid',
                                           activation=tf.keras.activations.relu,
                                           kernel_constraint=tf.keras.constraints.MaxNorm(max_value=3.))
    for kernel_size in [3, 4, 5]]
    self.pooling=tf.keras.layers.GlobalMaxPooling1D()
    self.dropout=tf.keras.layers.Dropout(kargs['dropout_rate'])
    self.fc1=tf.keras.layers.Dense(units=kargs['hidden_dimension'], activation=tf.keras.activations.relu,
                                   kernel_constraint=tf.keras.constraints.MaxNorm(max_value=3.))
    self.fc2=tf.keras.layers.Dense(units=kargs['output_dimension'], activation=tf.keras.activations.sigmoid,
                                   kernel_constraint=tf.keras.constraints.MaxNorm(max_value=3.))
  def call(self, x):
    x=self.embedding(x)
    x=self.dropout(x)
    x=tf.concat([self.pooling(conv(x)) for conv in self.conv_list], axis=-1)
    x=self.fc1(x)
    x=self.fc2(x)
    return x
```

모델 생성

```
model=CNNClassifier(**kargs)
model.compile(optimizer=tf.keras.optimizers.Adam(1e-4), loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy')])
```

모델 학습

```
import os
earlystop_callback=tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', min_delta=0.0001, patience=2)
checkpoint_path=DATA_IN_PATH+model_name+'/weight.h5'
checkpoint_dir=os.path.dirname(checkpoint_path)

if os.path.exists(checkpoint_dir):
  print("{} --Folder already exits \n".format(checkpoint_dir))
else:
  os.makedirs(checkpoint_dir, exist_ok=True)
  print("{} --Folder create complete \n".format(checkpoint_dir))

cp_callback=tf.keras.callbacks.ModelCheckpoint(checkpoint_path, monitor='val_accuracy', verbose=1, save_best_only=True, save_weights_only=True)
history=model.fit(train_input, train_label, batch_size=BATCH_SIZE, epochs=NUM_EPOCHS, validation_split=VALID_SPLIT,
                  callbacks=[earlystop_callback, cp_callback])
```

```
/content/drive/MyDrive/Kaggle/word2vec-nlp-tutorial/cnn_classifier_en --Folder already exits

Epoch 1/2
44/44 [==============================] - 112s 3s/step - loss: 0.6943 - accuracy: 0.4963 - val_loss: 0.6918 - val_accuracy: 0.5008

Epoch 00001: val_accuracy improved from -inf to 0.50080, saving model to /content/drive/MyDrive/Kaggle/word2vec-nlp-tutorial/cnn_classifier_en/we
Epoch 2/2
44/44 [==============================] - 110s 3s/step - loss: 0.6915 - accuracy: 0.5430 - val_loss: 0.6894 - val_accuracy: 0.6616

Epoch 00002: val_accuracy improved from 0.50080 to 0.66160, saving model to /content/drive/MyDrive/Kaggle/word2vec-nlp-tutorial/cnn_classifier_en
```

데이터 제출 결과 아래와 같다.

```
TEST_INPUT_DATA='test_input.npy'
TEST_ID_DATA='test_id.npy'

test_input=np.load(open(DATA_IN_PATH+TEST_INPUT_DATA, 'rb'))
test_input=tf.keras.preprocessing.sequence.pad_sequences(test_input, maxlen=test_input.shape[1])
SAVE_FILE_NM='weight.h5'
model.load_weights(os.path.join(DATA_IN_PATH, model_name, SAVE_FILE_NM))
```

```
predictions=model.predict(test_input, batch_size=BATCH_SIZE)
predictions=predictions.squeeze(-1)

test_id=np.load(open(DATA_IN_PATH+TEST_ID_DATA, 'rb'), allow_pickle=True)
if not os.path.exists(DATA_IN_PATH):
  os.makedirs(DATA_IN_PATH)
output=pd.DataFrame(data={"id":list(test_id), "sentiment":list(predictions)})
output.to_csv(DATA_IN_PATH+"movie_review_result_cnn.csv", index=False, quoting=3)
```

Name	Submitted	Wait time	Execution time	Score
movie_review_result_rnn.csv	just now	1 seconds	1 seconds	0.51596

Complete

Word2vec의 경우 뭔가 잘못되었는지 점수가 매우 낮게 나왔다. RNN과 CNN이 학습 시간이 매우 길고 구현이 어렵지만 성능이 월등이 높아야 하는데 뭔가 잘못 되었는지 생각보다 낮게 나왔다.

Ⅱ. 한글 텍스트 분류

https://github.com/e9t/nsmc

네이버 영화 리뷰 감정 분석을 하였고 위 링크에서 데이터를 다운 받을 수 있다. CNN을 이용하였다.

1. 데이터 분석

```
print('전체 학습 데이터의 개수: {}'.format(len(train_data)))
```

```
전체 학습 데이터의 개수: 150000
```

```
train_length=train_data['document'].astype(str).apply(len)
train_length.head()
```

```
0    19
1    33
2    17
3    29
4    61
Name: document, dtype: int64
```

```
print('리뷰 길이 최댓값 : {}'.format(np.max(train_length)))
print('리뷰 길이 최솟값 : {}'.format(np.min(train_length)))
print('리뷰 길이 평균값 : {:.2f}'.format(np.mean(train_length)))
print('리뷰 길이 표준편차 : {:.2f}'.format(np.std(train_length)))
print('리뷰 길이 중간값 : {}'.format(np.median(train_length)))
print('리뷰 길이 제1사분위 : {}'.format(np.percentile(train_length, 25)))
print('리뷰 길이 제3사분위 : {}'.format(np.percentile(train_length, 75)))
```

```
리뷰 길이 최댓값 : 158
리뷰 길이 최솟값 : 1
리뷰 길이 평균값 : 35.24
리뷰 길이 표준편차 : 29.58
리뷰 길이 중간값 : 27.0
리뷰 길이 제1사분위 : 16.0
리뷰 길이 제3사분위 : 42.0
```

```
train_review=[review for review in train_data['document'] if type(review) is str]
```

```
print('긍정 리뷰 개수 : {}'.format(train_data['label'].value_counts()[1]))
print('부정 리뷰 개수 : {}'.format(train_data['label'].value_counts()[0]))
```

```
긍정 리뷰 개수 : 74827
부정 리뷰 개수 : 75173
```

```
train_word_counts=train_data['document'].astype(str).apply(lambda x:len(x.split(' ')))
```

```
print('리뷰 단어 개수 최댓값 : {}'.format(np.max(train_word_counts)))
print('리뷰 단어 개수 최솟값 : {}'.format(np.min(train_word_counts)))
print('리뷰 단어 개수 평균값 : {:.2f}'.format(np.mean(train_word_counts)))
print('리뷰 단어 개수 표준편차 : {:.2f}'.format(np.std(train_word_counts)))
print('리뷰 단어 개수 중간값 : {}'.format(np.median(train_word_counts)))
print('리뷰 단어 개수 제1사분위 : {}'.format(np.percentile(train_word_counts, 25)))
print('리뷰 단어 개수 제3사분위 : {}'.format(np.percentile(train_word_counts, 75)))
```

```
리뷰 단어 개수 최댓값 : 41
리뷰 단어 개수 최솟값 : 1
리뷰 단어 개수 평균값 : 7.58
리뷰 단어 개수 표준편차 : 6.51
리뷰 단어 개수 중간값 : 6.0
리뷰 단어 개수 제1사분위 : 3.0
리뷰 단어 개수 제3사분위 : 9.0
```

```
qmarks=np.mean(train_data['document'].astype(str).apply(lambda x:'?' in x))
fullstop=np.mean(train_data['document'].astype(str).apply(lambda x:'.' in x))

print('물음표가 있는 리뷰: {:.2f}'.format(qmarks*100))
print('마침표가 있는 리뷰: {:.2f}'.format(fullstop*100))
```

```
물음표가 있는 리뷰: 8.25
마침표가 있는 리뷰: 51.76
```

2. 데이터 전처리

```
import numpy as np
import pandas as pd
import re
import json
from konlpy.tag import Okt
from tensorflow.python.keras.preprocessing.sequence import pad_sequences
from tensorflow.python.keras.preprocessing.text import Tokenizer

train_data=pd.read_csv(DATA_IN_PATH+'ratings_train.txt', header=0, delimiter='₩t', quoting=3)
```

불용어 제거

```
def preprocessing(review, okt, remove_stopwords=False, stop_words=[]):
  review_text=re.sub("[^가-힣ㄱ-ㅎㅏ-ㅣ\\s]", "", review)
  word_review=okt.morphs(review_text, stem=True)
  if remove_stopwords:
    word_review=[token for token in word_review if not token in stop_words]
  return word_review
```

```
stop_words=['은', '는', '이', '가', '하', '아', '것', '들', '의', '있', '되', '수', '보', '주', '등', '한']
okt=Okt()
clean_train_review=[]
for review in train_data['document']:
  if type(review)==str:
    clean_train_review.append(preprocessing(review, okt, remove_stopwords=True, stop_words=stop_words))
  else:
    clean_train_review.append([])
```

```
test_data=pd.read_csv(DATA_IN_PATH+'ratings_test.txt', header=0, delimiter='\t', quoting=3)
clean_test_review=[]
for review in test_data['document']:
  if type(review)==str:
    clean_test_review.append(preprocessing(review, okt, remove_stopwords=True, stop_words=stop_words))
  else:
    clean_test_review.append([])
```

데이터 벡터화, 단어 사전 저장

```
tokenizer=Tokenizer()
tokenizer.fit_on_texts(clean_train_review)
train_sequences=tokenizer.texts_to_sequences(clean_train_review)
test_sequences=tokenizer.texts_to_sequences(clean_test_review)

word_vocab=tokenizer.word_index #단어 사전 형태

MAX_SEQUENCE_LENGTH=8 #문장 최대 길이

#학습 데이터 벡터화
train_inputs=pad_sequences(train_sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post')
train_labels=np.array(train_data['label']) #학습 데이터 라벨
#평가 데이터 벡터화
test_inputs=pad_sequences(test_sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post')
#평가 데이터 라벨
test_labels=np.array(test_data['label'])
```

```
TRAIN_INPUT_DATA='nsmc_train_input.npy'
TRAIN_LABEL_DATA='nsmc_train_label.npy'
TEST_INPUT_DATA='nsmc_test_input.npy'
TEST_LABEL_DATA='nsmc_test_label.npy'
DATA_CONFIGS='data_configs.json'

data_configs={}

data_configs['vocab']=word_vocab
data_configs['vocab_size']=len(word_vocab)+1

np.save(open(DATA_IN_PATH+TRAIN_INPUT_DATA, 'wb'), train_inputs)
np.save(open(DATA_IN_PATH+TRAIN_LABEL_DATA, 'wb'), train_labels)
np.save(open(DATA_IN_PATH+TEST_INPUT_DATA, 'wb'), test_inputs)
np.save(open(DATA_IN_PATH+TEST_LABEL_DATA, 'wb'), test_labels)

json.dump(data_configs, open(DATA_IN_PATH+DATA_CONFIGS, 'w'), ensure_ascii=False)
```

3. 모델링

```
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras import layers

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import json

from tqdm import tqdm
```

```
INPUT_TRAIN_DATA='nsmc_train_input.npy'
LABEL_TRAIN_DATA='nsmc_train_label.npy'
DATA_CONFIGS='data_configs.json'

train_input=np.load(open(DATA_IN_PATH+INPUT_TRAIN_DATA, 'rb'))
train_input=pad_sequences(train_input, maxlen=train_input.shape[1])
train_label=np.load(open(DATA_IN_PATH+LABEL_TRAIN_DATA, 'rb'))
prepro_configs=json.load(open(DATA_IN_PATH+DATA_CONFIGS, 'r'))
```

파라미터 정의

```
#파라미터 정의
model_name='cnn_classifier_kr'
BATCH_SIZE=512
NUM_EPOCHS=2
VALID_SPLIT=0.1
MAX_LEN=train_input.shape[1]

kargs={'model_name':model_name, 'vocab_size':prepro_configs['vocab_size'], 'embedding_size':128, 'num_filters':100,
       'dropout_rate':0.5, 'hidden_dimension':250, 'output_dimension':1}
```

CNN 모델 함수

```
class CNNClassifier(tf.keras.Model):
  def __init__(self, **kargs):
    super(CNNClassifier, self).__init__(name=kargs['model_name'])
    self.embedding=tf.keras.layers.Embedding(input_dim=kargs['vocab_size'], output_dim=kargs['embedding_size'])
    self.conv_list=[tf.keras.layers.Conv1D(filters=kargs['num_filters'], kernel_size=kernel_size, padding='valid',
                                           activation=tf.keras.activations.relu,
                                           kernel_constraint=tf.keras.constraints.MaxNorm(max_value=3.))
    for kernel_size in [3, 4, 5]]
    self.pooling=tf.keras.layers.GlobalMaxPooling1D()
    self.dropout=tf.keras.layers.Dropout(kargs['dropout_rate'])
    self.fc1=tf.keras.layers.Dense(units=kargs['hidden_dimension'], activation=tf.keras.activations.relu,
                                   kernel_constraint=tf.keras.constraints.MaxNorm(max_value=3.))
    self.fc2=tf.keras.layers.Dense(units=kargs['output_dimension'], activation=tf.keras.activations.sigmoid,
                                   kernel_constraint=tf.keras.constraints.MaxNorm(max_value=3.))
  def call(self, x):
   x=self.embedding(x)
   x=self.dropout(x)
   x=tf.concat([self.pooling(conv(x)) for conv in self.conv_list], axis=-1)
   x=self.fc1(x)
   x=self.fc2(x)
   return x
```

```
model=CNNClassifier(**kargs)
model.compile(optimizer=tf.keras.optimizers.Adam(1e-4), loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy')])
```

4. 학습

```
import os
earlystop_callback=tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', min_delta=0.0001, patience=2)
checkpoint_path=DATA_IN_PATH+model_name+'/weight.h5'
checkpoint_dir=os.path.dirname(checkpoint_path)

if os.path.exists(checkpoint_dir):
  print("{} --Folder already exits \n".format(checkpoint_dir))
else:
  os.makedirs(checkpoint_dir, exist_ok=True)
  print("{} --Folder create complete \n".format(checkpoint_dir))

cp_callback=tf.keras.callbacks.ModelCheckpoint(checkpoint_path, monitor='val_accuracy',
                                               verbose=1, save_best_only=True, save_weights_only=True)
history=model.fit(train_input, train_label, batch_size=BATCH_SIZE, epochs=NUM_EPOCHS, validation_split=VALID_SPLIT,
                  callbacks=[earlystop_callback, cp_callback])
```

```
/content/drive/MyDrive/Kaggle/네이버 영화 리뷰 감정 분석/cnn_classifier_kr --Folder create complete

Epoch 1/2
264/264 [==============================] - 44s 156ms/step - loss: 0.6743 - accuracy: 0.5849 - val_loss: 0.4783 - val_

Epoch 00001: val_accuracy improved from -inf to 0.77620, saving model to /content/drive/MyDrive/Kaggle/네이버 영화 리
Epoch 2/2
264/264 [==============================] - 39s 149ms/step - loss: 0.4690 - accuracy: 0.7770 - val_loss: 0.4211 - val_

Epoch 00002: val_accuracy improved from 0.77620 to 0.80520, saving model to /content/drive/MyDrive/Kaggle/네이버 영화
```

```
INPUT_TEST_DATA='nsmc_test_input.npy'
LABEL_TEST_DATA='nsmc_test_label.npy'
SAVE_FILE_NM='weight.h5'

test_input=np.load(open(DATA_IN_PATH+INPUT_TEST_DATA, 'rb'))
test_input=pad_sequences(test_input, maxlen=test_input.shape[1])
test_label_data=np.load(open(DATA_IN_PATH+LABEL_TEST_DATA, 'rb'))
model.load_weights(os.path.join(DATA_IN_PATH, model_name, SAVE_FILE_NM))
model.evaluate(test_input, test_label_data)
```

```
1563/1563 [==============================] - 4s 3ms/step - loss: 0.4264 - accuracy: 0.8053
[0.42643046379089355, 0.8052600026130676]
```

정확도가 약 80% 나왔다.