In [ ]:

```
# 구글 코랩 서버에 KoNLPy 라이브러리 설치
!pip install konlpy
```

In [ ]:

```
from google.colab import drive
drive.mount('/gdrive', force_remount=True)
```

In [ ]:

```
# 데이터 파일 읽어오기
# ratings_train.txt: 15만개의 영화평, 긍정과 부정이 각각 50%씩
# ratings_test.txt: 5만개의 영화평, 긍정과 부정이 각각 50%씩

def read_data(filename):
    with open(filename, 'r') as f:
        data = [line.split('₩t') for line in f.read().splitlines()]
        # txt 파일의 헤더(id document label)는 제외하기
        data = data[1:]
    return data

train_data = read_data('/gdrive/My Drive/Keras/ratings_train.txt')
test_data = read_data('/gdrive/My Drive/Keras/ratings_test.txt')

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

In [ ]:

```
for i in range(5):
  print(train_data[i])
```

In [ ]:

```
for i in range(5):
  print(test_data[i])
```

In [ ]:

```
# 데이터 전처리 #1: KoNLPy 라이브러리를 이용하여 형태소 분석 및 품사 태깅
from konlpy.tag import Okt

okt = Okt()
# 문장 하나에 대해서 테스트
print(okt.pos(u'이 밤 그날의 반딧불을 당신의 창 가까이 보낼게요 ㅠㅠ'))
```

In [ ]:

```
# 훈련 데이터와 시험 데이터에 대한 형태소 분석 및 품사 태깅
# 데이터 양이 큰 만큼 시간이 오래 걸리기 때문에 형태소 분석 및 품사 태깅 작업을 마친 후 json 파일로 저장
# 이미 작업이 완료된 train_docs.json과 test_docs.json 파일이 존재하면 분석 작업은 건너뜀

import json
import os
from pprint import pprint

def tokenize(doc):
    # norm은 정규화, stem은 근어로 표시하기를 나타냄
    return ['/'.join(t) for t in okt.pos(doc, norm=True, stem=True)]

if os.path.isfile('/gdrive/My Drive/Keras/train_docs.json'):
    with open('/gdrive/My Drive/Keras/train_docs.json') as f:
        print('Train json file loading....')
        train_docs = json.load(f)
    with open('/gdrive/My Drive/Keras/test_docs.json') as f:
        print('Test json file loading....')
        test_docs = json.load(f)
else:
    train_docs = [(tokenize(row[1]), row[2]) for row in train_data]
    test_docs = [(tokenize(row[1]), row[2]) for row in test_data]
    # JSON 파일로 저장
    with open('/gdrive/My Drive/Keras/train_docs.json', 'w', encoding="utf-8") as make_file:
        json.dump(train_docs, make_file, ensure_ascii=False, indent="₩t")
    with open('/gdrive/My Drive/Keras/test_docs.json', 'w', encoding="utf-8") as make_file:
        json.dump(test_docs, make_file, ensure_ascii=False, indent="₩t")

# 예쁘게 출력하기 위해서 pprint 라이브러리 사용
pprint(train_docs[0])
```

In [ ]:

```
# 분석한 데이터의 토큰(문자열 분석을 위한 작은 단위)의 개수 확인
tokens = [t for d in train_docs for t in d[0]]
print(len(tokens))
```

In [ ]:

```
tokens
```

In [ ]:

```
# 데이터 전처리 #2: nltk 라이브러리 활용하여 데이터 벡터화

# Text 클래스는 문서 분석에 유용한 다양한 기능 제공
import nltk
text = nltk.Text(tokens, name='NMSC')

# 전체 토큰의 개수
print(len(text.tokens))

# 중복을 제외한 토큰의 개수
print(len(set(text.tokens)))

# 출현 빈도가 높은 상위 토큰 10개
pprint(text.vocab().most_common(10))
```

In [ ]:

```
# 자주 사용되는 토큰 2,000개를 사용하여 데이터를 벡터화 시킴
# CountVectorization 사용: 문서 집합에서 단어 토큰을 생성하고 각 단어의 수를 세어
# BOW(Bag of Words) 인코딩한 벡터를 만드는 역할

# 시간이 꽤 걸립니다! 시간을 절약하고 싶으면 most_common의 매개변수를 줄여보세요.
# 원래는 10,000개 였으나 시간이 오래 걸리고 메모리 오류 발생으로 2,000개로 줄임

selected_words = [f[0] for f in text.vocab().most_common(2000)]

def term_frequency(doc):
    return [doc.count(word) for word in selected_words]

train_x = [term_frequency(d) for d, _ in train_docs]
test_x = [term_frequency(d) for d, _ in test_docs]
train_y = [c for _, c in train_docs]
test_y = [c for _, c in test_docs]
```

In [ ]:

```
len(train_x[0]), len(train_y[0])
```

In [ ]:

```
# 데이터 전처리 #3: 데이터를 numpy 배열에 저장하고, 데이터를 float로 형 변환
import numpy as np

x_train = np.asarray(train_x).astype('float32')
x_test = np.asarray(test_x).astype('float32')

y_train = np.asarray(train_y).astype('float32')
y_test = np.asarray(test_y).astype('float32')
```

In [ ]:

```
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

In [ ]:

```
# 모델 정의
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import RMSprop

model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(2000,)))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

In [ ]:

```
 # 학습하기
model.compile(optimizer=RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=10, batch_size=512)
```

In [ ]:

```
# 모델 평가하기
results = model.evaluate(x_test, y_test)
results
```

In [ ]:

```
# 새로운 데이터로 결과 예측하기

def predict_pos_neg(review):
    token = tokenize(review)
    tf = term_frequency(token)
    data = np.expand_dims(np.asarray(tf).astype('float32'), axis=0)
    score = float(model.predict(data))
    if(score > 0.5):
        print("[{}]는 {:.2f}% 확률로 긍정 리뷰이지 않을까 추측해봅니다.^^₩n".format(review, sco
re * 100))
    else:
        print("[{}]는 {:.2f}% 확률로 부정 리뷰이지 않을까 추측해봅니다.^^;₩n".format(review, (1
- score) * 100))

predict_pos_neg("올해 최고의 영화! 세 번 넘게 봐도 질리지가 않네요.")
predict_pos_neg("배경 음악이 영화의 분위기랑 너무 안 맞았습니다. 몰입에 방해가 됩니다.")
predict_pos_neg("주연 배우가 신인인데 연기를 진짜 잘 하네요. 몰입감 ㅎㄷㄷ")
predict_pos_neg("믿고 보는 감독이지만 이번에는 아니네요")
predict_pos_neg("주연배우 때문에 봤어요")
predict_pos_neg("남자 주인공이 너무 잘 생겼어요.")
```

In [ ]:

```
# selected_words : 토큰들 중에서 자주 사용되는 단어 2,000개가 저장된 리스트

idx = 0
for w in selected_words:
  print(str(idx) + ":" + w)
  idx += 1
```

In [ ]:

```
# 첫번째 훈련 데이터: x_train[0]
# [['아/Exclamation',
#  '더빙/Noun',
#  '../Punctuation',
#  '진짜/Noun',
#  '짜증나다/Adjective',
#  '목소리/Noun'],
# '0']

# 각각의 단어들이 selected_words 리스트에서 인덱스값으로 변환되어 저장되어 있음

idx = 0
for value in x_train[0]:
  if( value == 1.0 ):
    print(idx)
  idx += 1
```