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

# 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

from konlpy.tag import Okt
import json
import os
import re

okt = Okt()

def tokenize(doc):
    # 특수 문자 제거
    doc = re.sub(r"[-()₩"#/@;:<>{}`+=~|.!?,]", "", doc)
    # norm은 정규화, stem은 근어로 표시하기를 나타냄
    tmp = ['/'.join(t) for t in okt.pos(doc, norm=True, stem=True)]
    #print(tmp)
    ret_tokenize = []
    for token in tmp:
      (word, kind) = token.split('/')
      # 조사는 제외시킴
      if(kind != "Josa"):
        ret_tokenize.append(word)

    return ret_tokenize

if os.path.isfile('/gdrive/My Drive/Keras/train_docs2.json'):
    with open('/gdrive/My Drive/Keras/train_docs2.json') as f:
        print('Train json file loading....')
        train_docs = json.load(f)
    with open('/gdrive/My Drive/Keras/test_docs2.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_docs2.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_docs2.json', 'w', encoding="utf-8") as make_file:
        json.dump(test_docs, make_file, ensure_ascii=False, indent="₩t")
```

In [ ]:

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

In [ ]:

```
# 데이터 전처리 #2: 데이터 벡터화
from tensorflow.keras.preprocessing.text import Tokenizer

train_x = [token for token, _ in train_docs]
test_x = [token for token, _ in test_docs]

max_words = 10000
tokenizer = Tokenizer(num_words = max_words) # 상위 10,000개의 단어만 보존
tokenizer.fit_on_texts(train_x)
train_x = tokenizer.texts_to_sequences(train_x)
test_x = tokenizer.texts_to_sequences(test_x)
```

In [ ]:

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

In [ ]:

```
# 가장 긴 리뷰의 길이가 약 70정도이고, 리뷰의 평균 길이는 10정도 됨
# 전체 데이터의 길이는 30으로 맞춤

from tensorflow.keras.preprocessing.sequence import pad_sequences

max_len = 30

train_x = pad_sequences(train_x, maxlen = max_len)
test_x = pad_sequences(test_x, maxlen = max_len)
```

In [ ]:

```
# y값 추출

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 [ ]:

```
train_x[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 Embedding, Dense, LSTM
from keras.optimizers import RMSprop

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

model.summary()
```

In [ ]:

```
 # 학습하기 - 1에폭당 1분 50초정도 걸림

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 [ ]:

```
print(x_test[0])
```

In [ ]:

```
# 새로운 데이터로 결과 예측하기 - 수정 필요!!!!!

def predict_pos_neg(review):
    token = [tokenize(review)]
    #print(token)
    token = tokenizer.texts_to_sequences(token)
    token = pad_sequences(token, maxlen = max_len)
    #print(token)
    token = np.asarray(token).astype('float32')
    score = float(model.predict(token))
    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.")
```