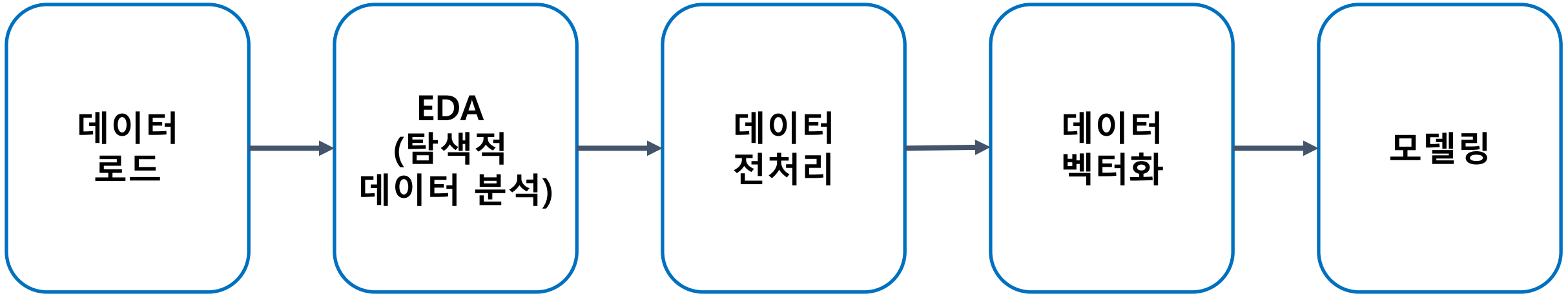
텍스트 분류


ІНСТИТУТ МІСТА
ГТЦ №OEZ121
УГОДИ №26-52
ГТЦ № OEZ 250
УГОДИ №1-2
ГТЦ № OEZ 260
OEZ 331
ГТЦ №OEZ172
УГОДИ №68-90
ГТЦ №OEZ172
УГОДИ №91-108
OEZ 314
OEZ 316
OEZ 285
OEZ 280
OEZ 409 (2017)
ГТЦ №OEZ131
УГОДИ №1-25
IV ФМС-2016
V ФМС-2017
OEZ 338 (ММ)
OEZ 342 (ММ)
МЕРМ
МАТРА

박경규

텍스트 분류 절차

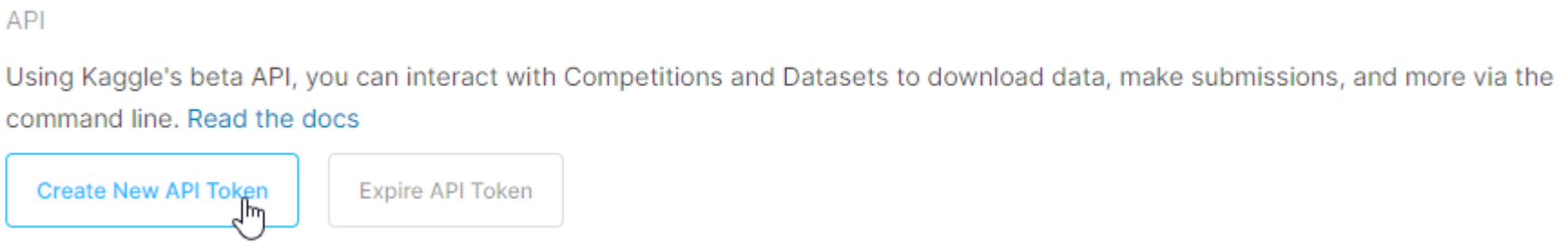
자연어 처리 기술을 활용해 문장의 정보를 추출해서 사람이 정한 범주(Class)로 분류하는 문제입니다.



IMDB 리뷰 감성 분류

■ kagge 가입 및 kgggle.json 다운로드

- kaggle 가입 : https://www.kaggle.com/
- kgggle.json 다운로드 : https://www.kaggle.com/<username>/account



■ 데이터 필드

- 데이터 : IMDB movie reviews, https://www.kaggle.com/c/word2vec-nlp-tutorial
- 데이터 필드 : id, sentiment(1 - positive review, 0 - negative review), review

IMDB 리뷰 감성 분류

■ kaggle 패키지 설치

```
! pip install  kaggle
```

■ kaggle API 키 업로드

```
from google.colab import files

files.upload()
```

```
! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json
```

■ kaggle 패키지 설치

```
! kaggle competitions download -c word2vec-nlp-tutorial
```

IMDB 리뷰 감성 분류

■ 압축 파일 해제

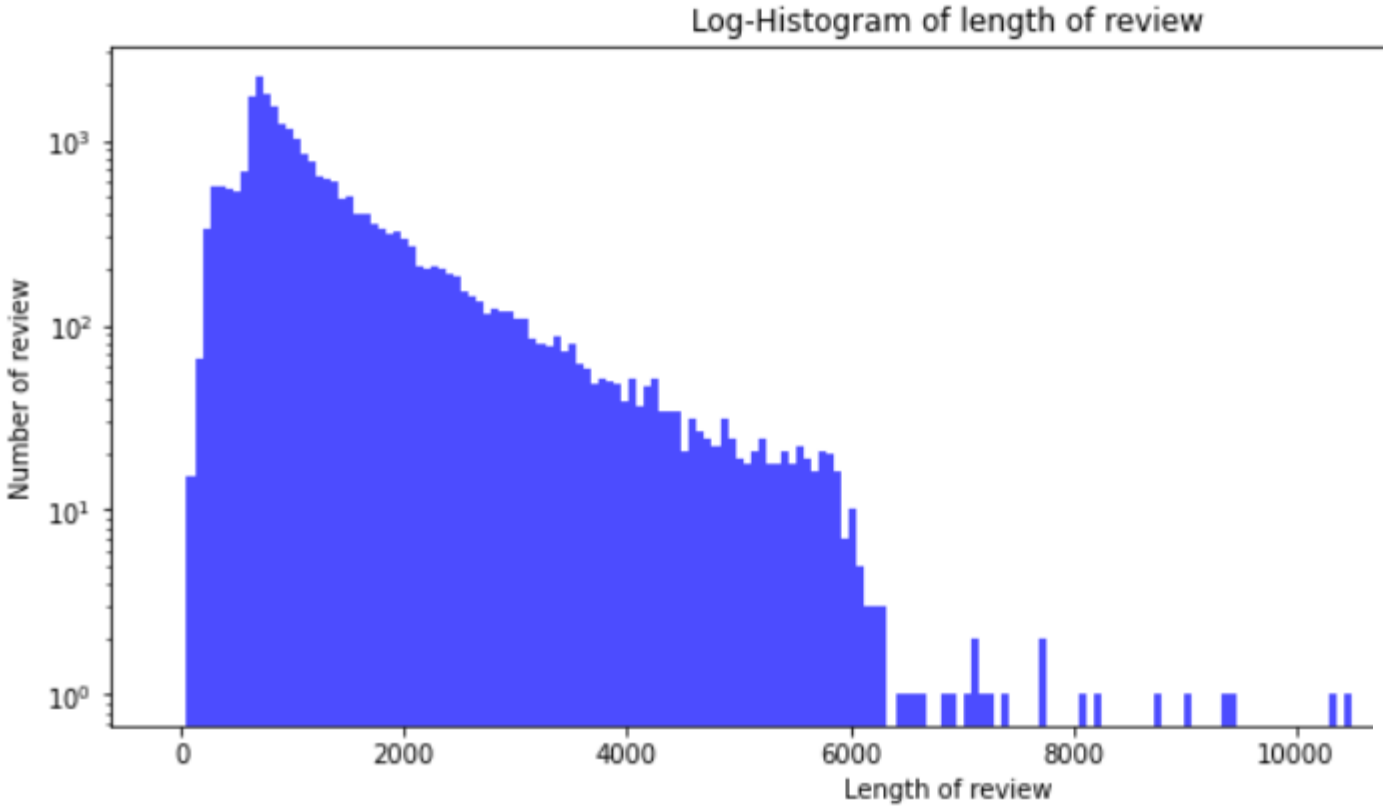
```
! unzip unlabeledTrainData.tsv.zip
! unzip labeledTrainData.tsv.zip
! unzip testData.tsv.zip
```

■ 데이터 로드

```
train_data = pd.read_csv('labeledTrainData.tsv', header=0,
delimiter='\t', quoting=3)

train_length = train_data['review'].apply(len)
```

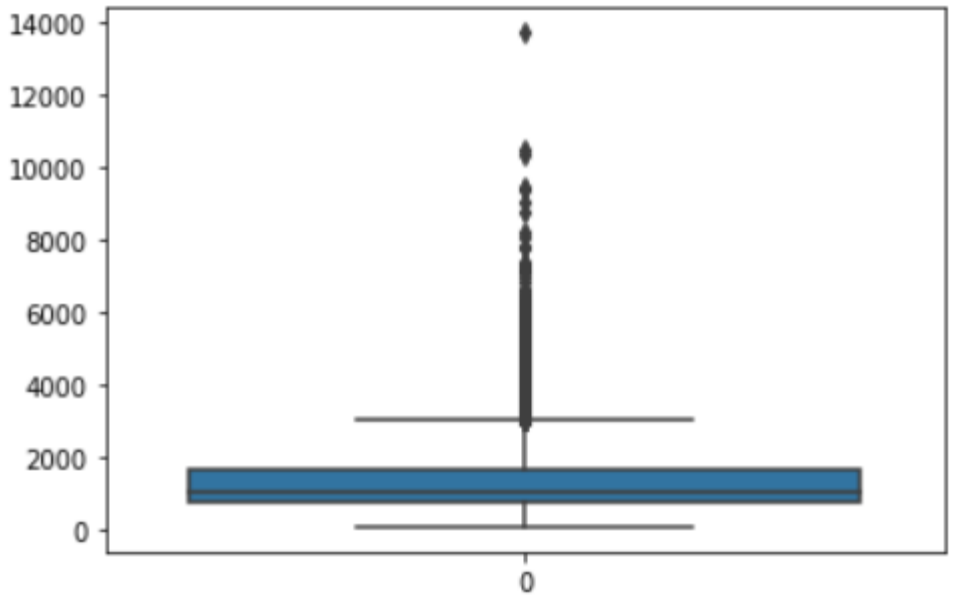
IMDB 리뷰 감성 분류 - 데이터 분석

■ 리뷰의 문자 길이 분포 분석



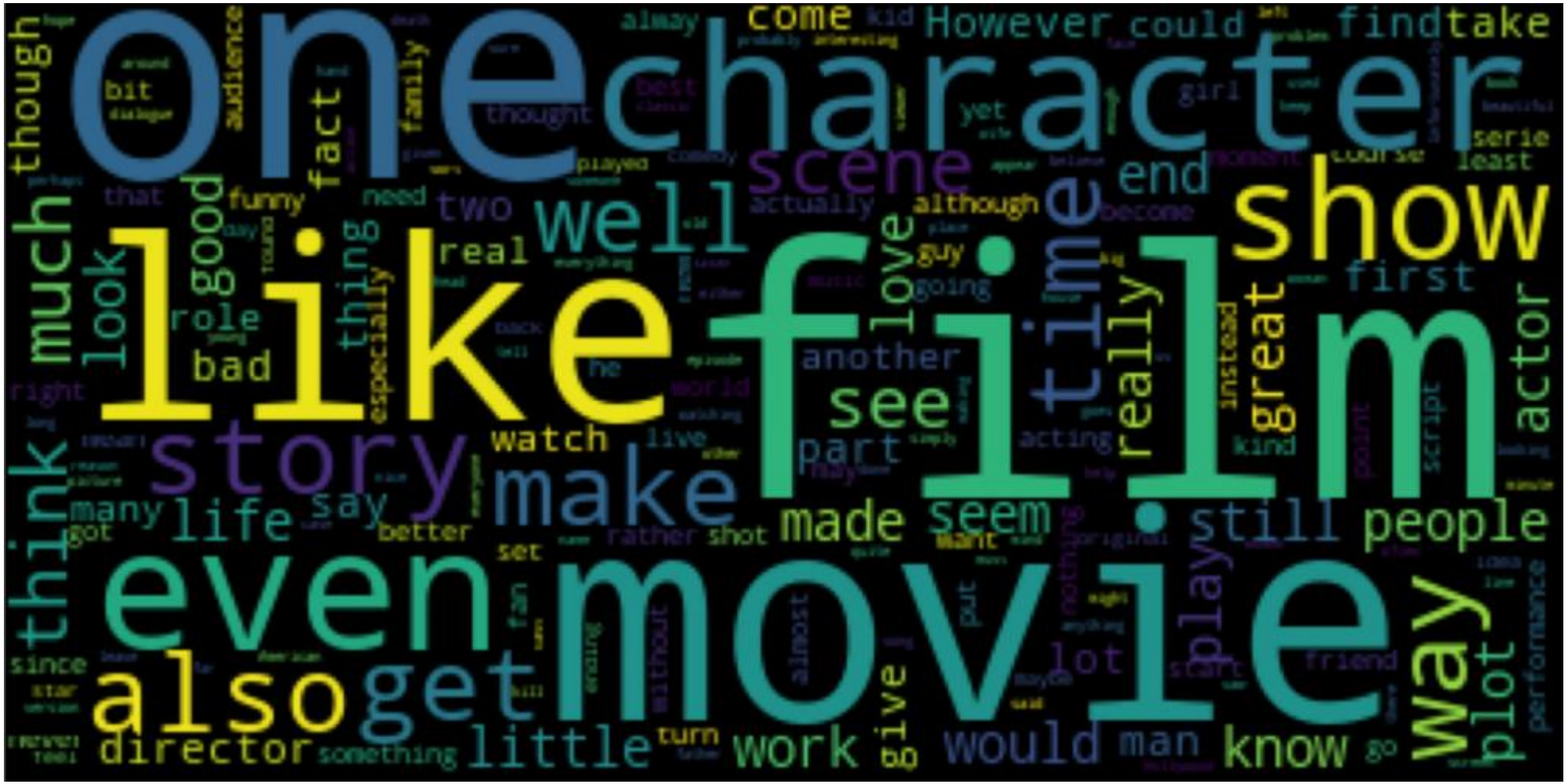
데이터 개수: 25000

리뷰 길이 최대 값: 13710
리뷰 길이 최소 값: 54
리뷰 길이 평균 값: 1329.71
리뷰 길이 표준편차: 1005.22
리뷰 길이 중간값: 983.0
리뷰 길이 제 1 사분위: 705.0
리뷰 길이 제 3 사분위: 1619.0



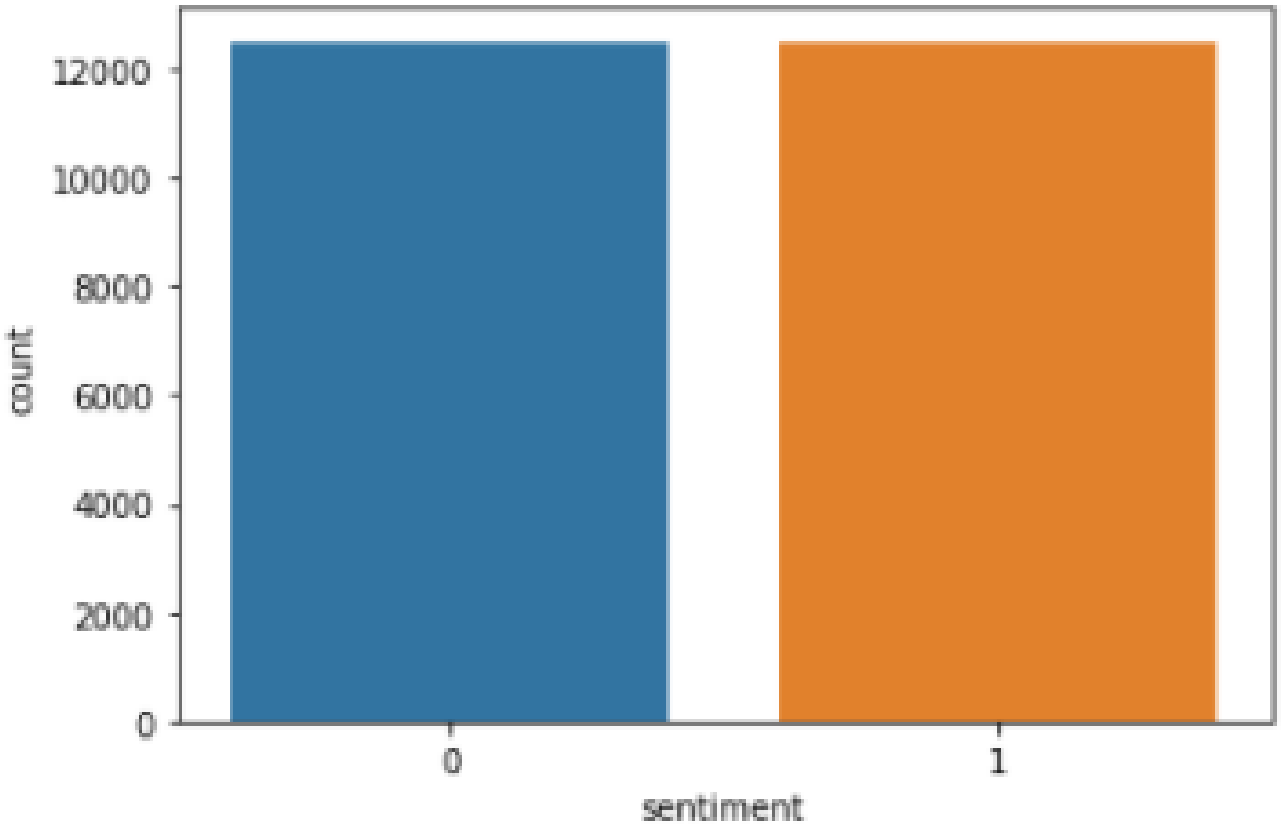
IMDB 리뷰 감성 분류 – 데이터 분석

■ 많이 사용된 단어 – Word Cloud



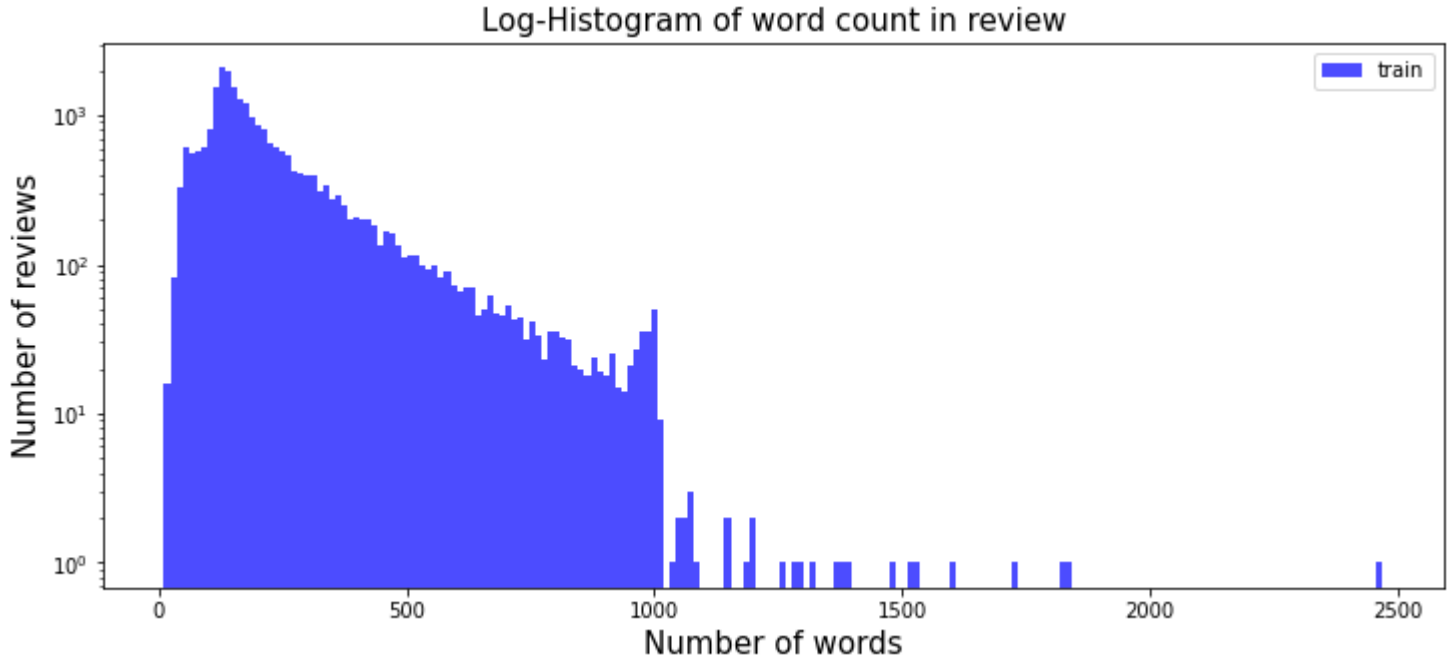
IMDB 리뷰 감성 분류 - 데이터 분석

■ 긍정, 부정 데이터(lable) 분포 확인



IMDB 리뷰 감성 분류 - 데이터 분석

■ 리뷰의 문자 길이 분포 분석



리뷰 단어 개수 최대 값: 2470
리뷰 단어 개수 최소 값: 10
리뷰 단어 개수 평균 값: 233.79
리뷰 단어 개수 표준편차: 173.74
리뷰 단어 개수 중간값: 174.0
리뷰 단어 개수 제 1 사분위: 127.0
리뷰 단어 개수 제 3 사분위: 284.0

IMDB 리뷰 감성 분류 - 데이터 전처리

■ 데이터 전처리 함수

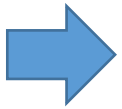
- HTML 태그 제거, 영어가 아닌 특수문자들을 공백으로 바꾸기
- 대문자들을 소문자로 바꾸기, 불용어 제거

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

clean_train_data = []
for review in train_data['review']:
    clean_train_data.append(preprocessing(review))
```

IMDB 리뷰 감성 분류 - 데이터 전처리

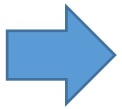
■ 단어를 인덱스로 벡터화

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

vocab = tokenizer.word_index
vocab["<PAD>"] = 0
print("전체 단어 개수: ", len(vocab))
```



```
전체 단어 개수:  74066
```

```
MAX_SEQUENCE_LENGTH = 200
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, 200)
```

IMDB 리뷰 감성 분류 - 데이터 전처리

■ 단어를 인덱스로 벡터화

```
classic war worlds timothy hines entertaining film obviously goes great effort lengths faithfully
[232, 203, 3048, 3565, 7116, 317, 2, 405, 153, 19, 634, 10967, 11898, 8816, 1653, 1035, 3494, 232,

classic -> 232
war -> 203
worlds -> 3048

문장 길이 : 84
인풋 길이 : 200
[  232   203  3048  3565  7116   317     2   405   153    19   634 10967
 11898  8816  1653  1035  3494   232   154   314  7116  2701   178     2
  2349    87  1111   582   217  2219   149    73   160   626  1035  2882
   194   642  3316  3464  3869   154   405   180   155   158    79     1
 19718  2177  1251    68  6828   170   281   811     1   532 10968     4
  2003    16    36   881  1251   376   634   314  7116   159  2564  1653
  1035  3494   232   511   143   317    20   623  4628  1251  8974  5471
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0]
```

IMDB 리뷰 감성 분류 - 데이터 전처리

■ 전처리 데이터 저장

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

DATA_PATH = '/content/drive/MyDrive/nlpdata/imdb/'
TRAIN_DATA = 'train_input.npy'
TRAIN_LABEL = 'train_label.npy'
TRAIN_CLEAN_DATA = 'train_clean.csv'
DATA_CONFIGS = 'data_configs.json'

# 전처리 된 데이터를 넘파이로 저장
np.save(open(DATA_PATH + TRAIN_DATA, 'wb'), train_inputs)
np.save(open(DATA_PATH + TRAIN_LABEL, 'wb'), train_labels)

# 정제된 텍스트를 csv로 저장
clean_train_df.to_csv(DATA_PATH + TRAIN_CLEAN_DATA, index=False)

# 데이터 사전을 json으로 저장
json.dump(data_configs, open(DATA_PATH + DATA_CONFIGS, 'w'), ensure_ascii=False)
```

IMDB 리뷰 감성 분류 – TF-IDF 활용 로지스틱 회귀모델

```
[1]  import os
     import numpy as np
     import pandas as pd

     from sklearn.feature_extraction.text import TfidfVectorizer
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LogisticRegression
```

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

```
     Mounted at /content/drive
```

TF-IDF 값으로 벡터화를 진행하므로 텍스트 데이터(train_clean.csv) 사용

```
[3]  DATA_PATH = '/content/drive/MyDrive/nlpdata/imdb/'
     TRAIN_CLEAN_DATA = 'train_clean.csv'
```

```
[4]  train_data = pd.read_csv( DATA_PATH + TRAIN_CLEAN_DATA )
```

```
[5]  reviews = list(train_data['review'])
     sentiments = list(train_data['sentiment'])
```

IMDB 리뷰 감성 분류 - TF-IDF 활용 로지스틱 회귀모델

```
[6]  vectorizer = TfidfVectorizer(min_df = 0.0, analyzer="char", sublinear_tf=True, ngram_range=(1,3), max_features=5000)

     X = vectorizer.fit_transform(reviews)
     y = np.array(sentiments)
```

```
[7]  features = vectorizer.get_feature_names()
```

학습과 검증 데이터셋 분리

```
[8]  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

로지스틱 회귀 모델 선언 및 학습

```
[9]  lr = LogisticRegression(class_weight='balanced')
     lr.fit(X_train, y_train)
```

```
[10] predicted = lr.predict(X_test)
     print(f"Accuracy: {lr.score(X_test, y_test):.2f}")
```

```
     Accuracy: 0.86
```

IMDB 리뷰 감성 분류 – TF-IDF 활용 로지스틱 회귀모델

```
[11]  TEST_CLEAN_DATA = 'test_clean.csv'

      test_data = pd.read_csv(DATA_PATH + TEST_CLEAN_DATA)
      testDataVecs = vectorizer.transform(test_data['review'])
      test_predicted = lr.predict(testDataVecs)
      print(test_predicted)
```

```
      [1 0 1 ... 0 1 0]
```

```
[12]  answer_dataset = pd.DataFrame({'id': test_data['id'], 'sentiment': test_predicted})
      answer_dataset.to_csv(DATA_PATH + 'answer_lr_tfidf.csv', index=False, quoting=3)
```

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

IMDB 리뷰 감성 분류 – Word2Vec을 활용 로지스틱 회귀모델

라이브러리 임포트

```
[1]  import logging
     import numpy as np
     import pandas as pd
     from gensim.models import word2vec
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LogisticRegression

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

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

```
     Mounted at /content/drive
```

```
[3]  DATA_PATH = '/content/drive/MyDrive/nlpdata/imdb/'
     TRAIN_CLEAN_DATA = 'train_clean.csv'
```

```
[4]  train_data = pd.read_csv(DATA_PATH + TRAIN_CLEAN_DATA)
```

```
[5]  reviews = list(train_data['review'])
     sentiments = list(train_data['sentiment'])
```

IMDB 리뷰 감성 분류 – Word2Vec을 활용 로지스틱 회귀모델

word2vev을 사용하기 위해 입력값을 단어로 구분된 리스트로 만듬

```
[6]  sentences = []
     for review in reviews:
         sentences.append(review.split())
```

word2vec 학습 및 모델 저장

- gensim 라이브러리가 설치 안 되어 있으면 설치
- `!pip install gensim`

```
[7]  num_features = 300      # 단어 임베딩 벡터 차원수
     min_word_count = 40     # 적은 빈도의 수의 단어는 학습하지 않음
     num_workers = 4         # 모델 학습 프로세스 개수
     context = 10            # 컨텍스트 윈도 크기
     downsampling = 1e-3     # 다운 샘플링 비율(보통 0.001)

     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(DATA_PATH +model_name)
```

IMDB 리뷰 감성 분류 – Word2Vec을 활용 로지스틱 회귀모델

분류 모델 학습을 위해 입력값을 같은 형태로 만듬

각 리뷰에 있는 전체 단어의 평균값을 계산하는 함수

- words : 단어의 모음인 하나의 리뷰 데이터
- model : 학습된 word2vec 모델
- num_features : 임베딩 벡터 차원수 index2word_set 셋으로 문장의 단어가 모델 단어사전에 속하는지 확인

1. model.wv.index2word로 set 객체 생성
2. 반복문으로 임베딩된 벡터가 있는 단어 벡터의 합을 구함
3. 단어의 전체 개수로 나누어 평균 벡터의 값 계산

```
[8]  def get_features(words, model, num_features):
         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
```

IMDB 리뷰 감성 분류 – Word2Vec을 활용 로지스틱 회귀모델

전체 리뷰에 대해 각 리뷰의 평균 벡터를 구하는 함수

```
[9]  def get_dataset(reviews, model, num_features):
         dataset = list()

         for s in reviews:
             dataset.append(get_features(s, model, num_features))

         reviewFeatureVecs = np.stack(dataset)

         return reviewFeatureVecs
```

```
[10]  test_data_vecs = get_dataset(sentences, model, num_features)
```

학습과 검증 데이터셋 분리

```
[11]  X = test_data_vecs
      y = np.array(sentiments)
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

IMDB 리뷰 감성 분류 – Word2Vec을 활용 로지스틱 회귀모델

로지스틱 회귀 모델 선언 및 학습

```
[12]  lr = LogisticRegression(class_weight='balanced')
      lr.fit(X_train, y_train)
      print(f"Accuracy: {lr.score(X_test, y_test):.2f}")
```

```
[13]  TEST_CLEAN_DATA = 'test_clean.csv'
      test_data = pd.read_csv(DATA_PATH + TEST_CLEAN_DATA)
      test_review = list(test_data['review'])
```

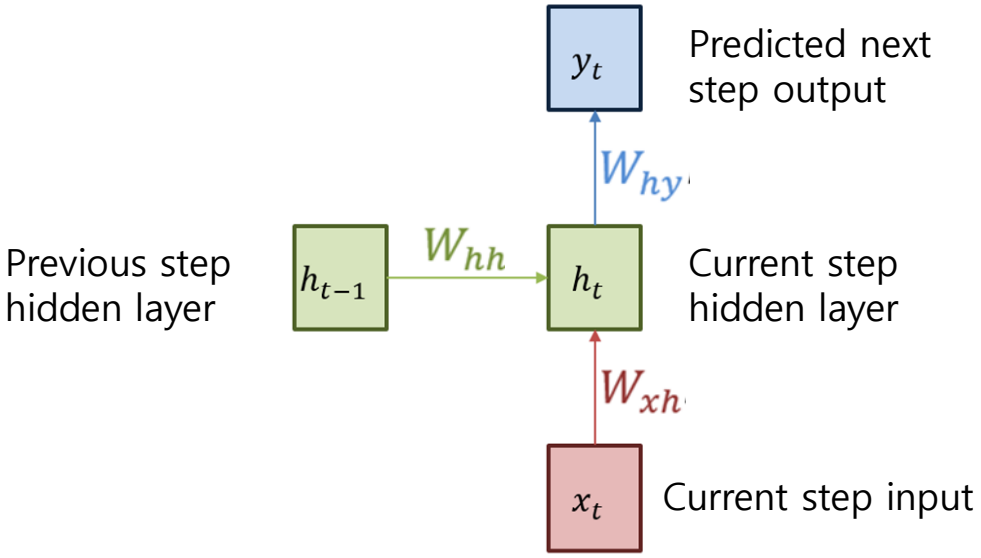
```
[14]  test_sentences = list()
      for review in test_review:
          test_sentences.append(review.split())
```

```
[15]  test_data_vecs = get_dataset(test_sentences, model, num_features)
```

```
[16]  ids = list(test_data['id'])
      test_predicted = lr.predict(test_data_vecs)

      answer_dataset = pd.DataFrame({'id': ids, 'sentiment': test_predicted})
      answer_dataset.to_csv(DATA_PATH + 'answer_lr_w2v.csv', index=False, quoting=3)
```

IMDB 리뷰 감성 분류 - RNN 모델

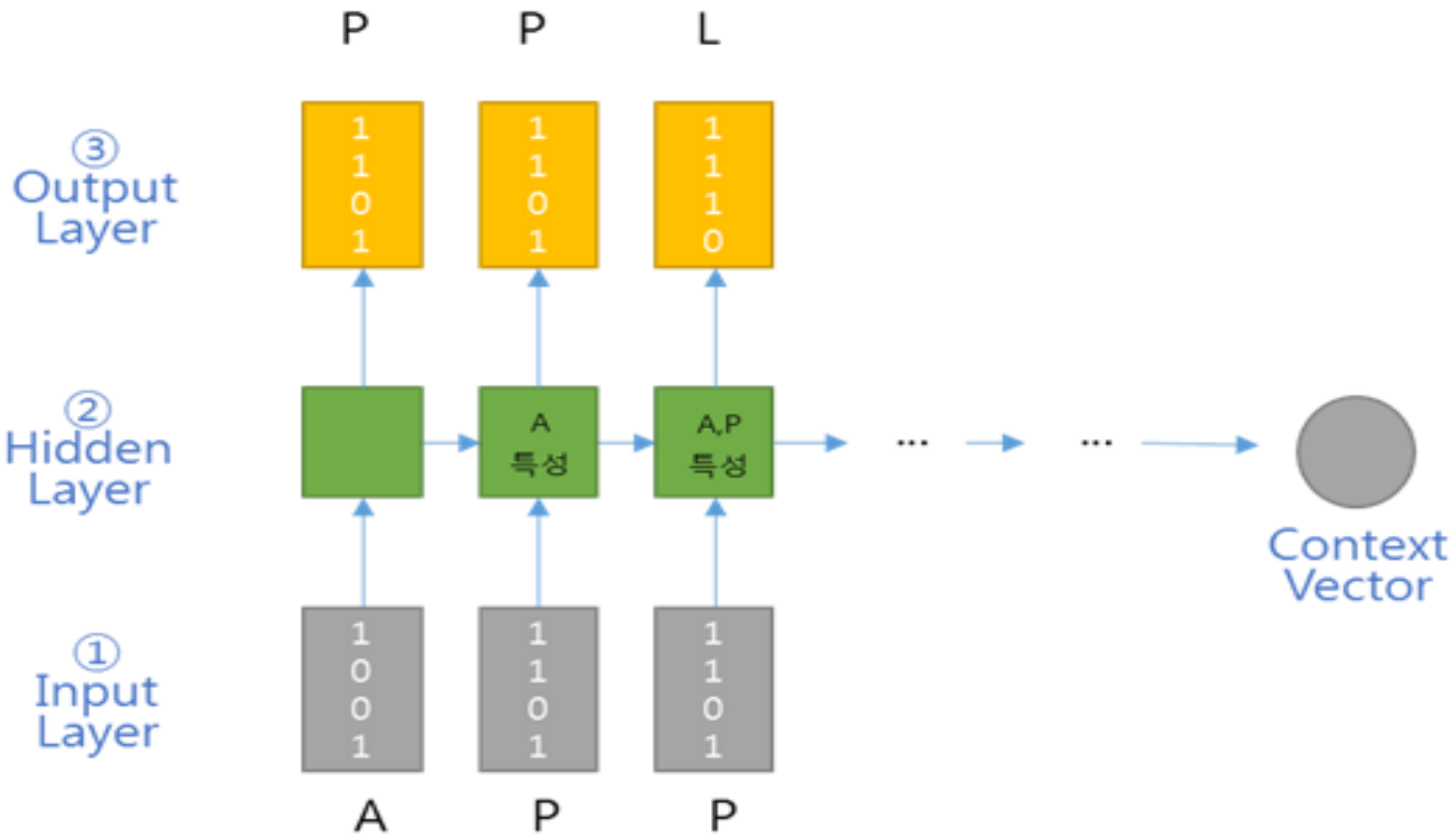
- RNN은 순환구조 인공신경망으로 이전 state 정보가 다음 state 를 예측하는데 사용되어 시계열 데이터 처리에 특화



$$y_t = W_{hy}h_t + b_y$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

IMDB 리뷰 감성 분류 - RNN 모델

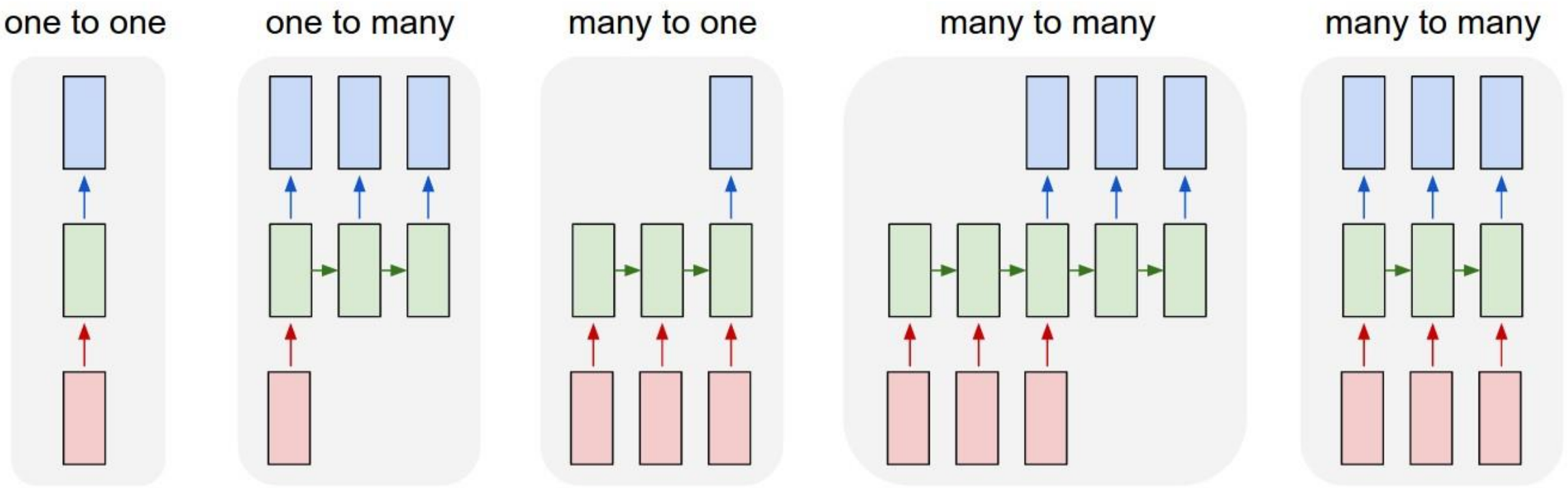


① Input Layer : 현재의 값
② Hidden Layer : 다음을 예측하기 위해 이전 값의 특성을 담는 곳
③ Output Layer : 현재를 기반으로 예측된 다음의 값

IMDB 리뷰 감성 분류 - RNN 모델

예. **감성 분류**
일련의 단어들 -> 감성

예. **기계 번역**
일련의 단어들 -> 일련의 단어들



예) 이미지 Captioning
이미지 → 일련의 단어들

예) 감성 분류
일련의 단어들 → 감성

예) 기계 번역
일련의 단어들 → 일련의 단어들

예) 프레임 수준에서 비디오 분류

IMDB 리뷰 감성 분류 - RNN 모델

데이터 로드

```
[3]  DATA_PATH = '/content/drive/MyDrive/nlpdata/imdb/'
     TRAIN_INPUT_DATA = 'train_input.npy'
     TRAIN_LABEL_DATA = 'train_label.npy'
     DATA_CONFIGS = 'data_configs.json'
```

```
[4]  train_input = np.load(open(DATA_PATH + TRAIN_INPUT_DATA, 'rb'))
     train_label = np.load(open(DATA_PATH + TRAIN_LABEL_DATA, 'rb'))
     prepro_configs = json.load(open(DATA_PATH + DATA_CONFIGS, 'r'))
```

모델 하이퍼파라메터 정의

```
[5]  BATCH_SIZE = 128
     NUM_EPOCHS = 5
     VALID_SPLIT = 0.2
     MAX_LEN = train_input.shape[1]

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

IMDB 리뷰 감성 분류 - RNN 모델

모델 구현

```
[6]  class RNNClassifier(tf.keras.Model):
         def __init__(self, **kargs):
             super(RNNClassifier, self).__init__(name=kargs['model_name'])
             self.embedding = 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 = layers.Dropout(kargs['dropout_rate'])
             self.fc1 = layers.Dense(units=kargs['dense_dimension'],
                                     activation=tf.keras.activations.tanh)
             self.fc2 = 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
```

IMDB 리뷰 감성 분류 - RNN 모델

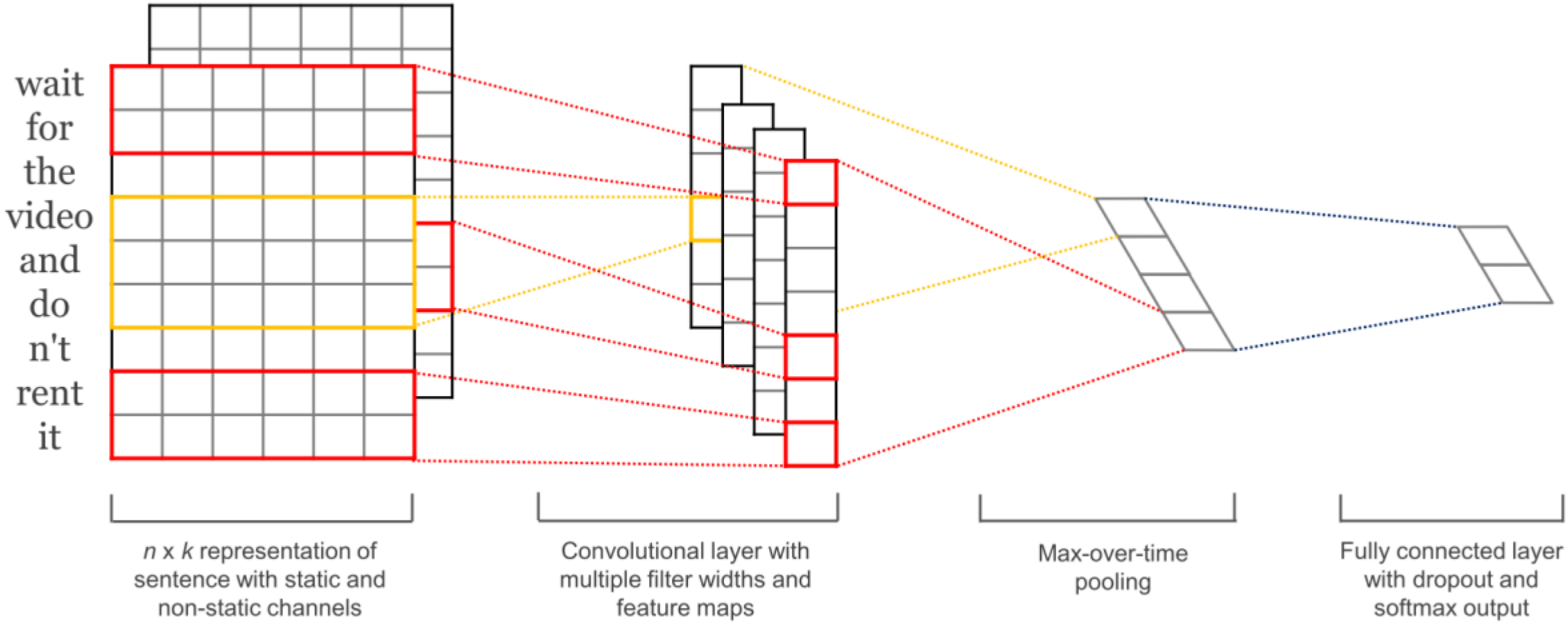
```
[7]  model = RNNClassifier(**kargs)
     model.compile(optimizer=tf.keras.optimizers.Adam(1e-4),
                   loss=tf.keras.losses.BinaryCrossentropy(),
                   metrics=['accuracy'])
```

```
[8]  checkpoint_path = DATA_PATH + '/weights_rnn.h5'
     # min_delta: the threshold that triggers the termination (acc should at least improve 0.0001)
     # patience: no improvment epochs (patience = 1, 1번 이상 개선이 없으면 종료)
     es = EarlyStopping(monitor='val_accuracy', min_delta=0.0001, patience=1)
     mc = ModelCheckpoint(checkpoint_path, monitor='val_accuracy', verbose=1,
                          save_best_only=True, save_weights_only=True)
```

```
[9]  history = model.fit(train_input, train_label, batch_size=BATCH_SIZE, epochs=NUM_EPOCHS,
                         validation_split=VALID_SPLIT, callbacks=[es, mc])
```

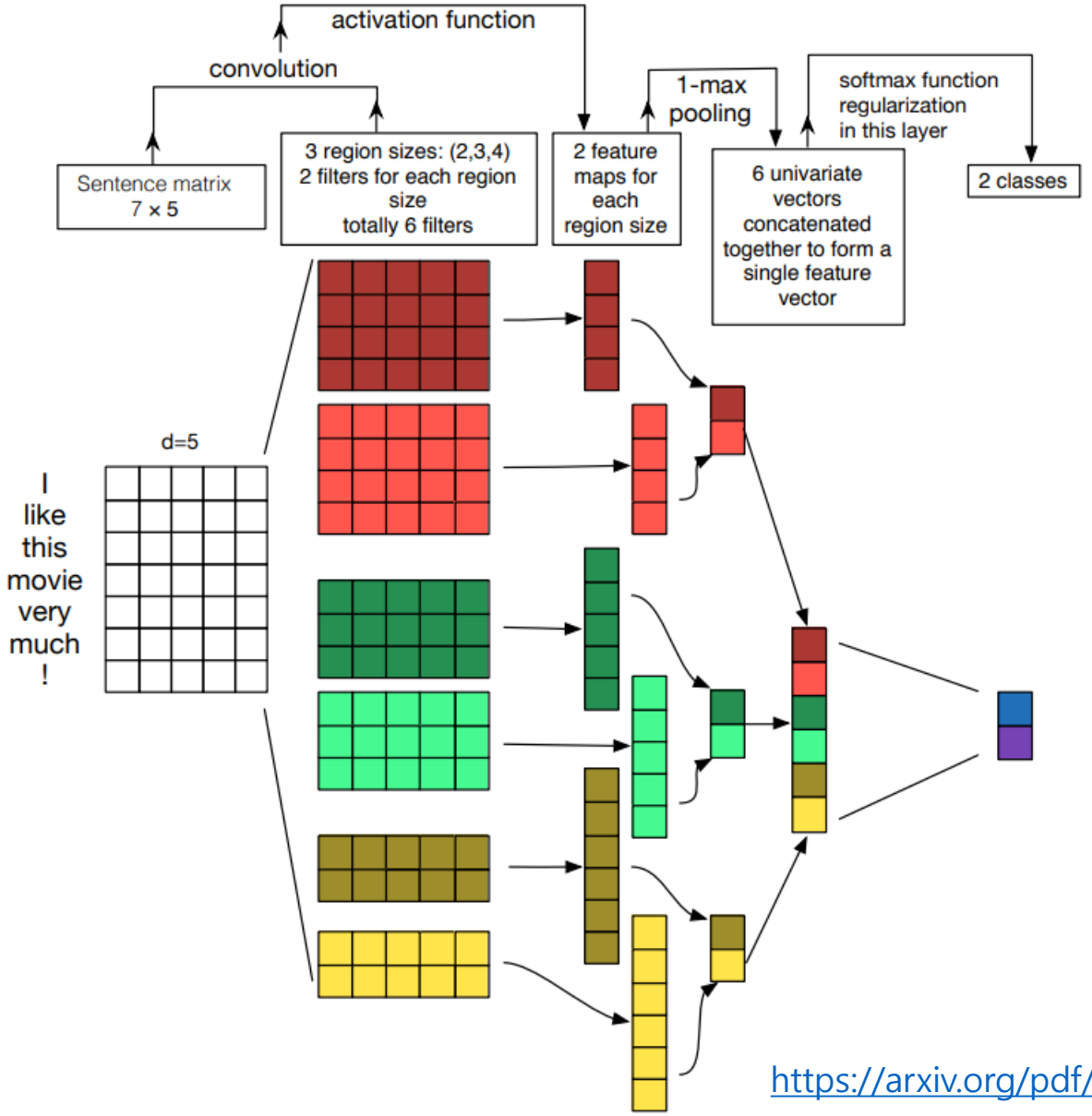
```
Epoch 1/5
157/157 [==============================] - 51s 109ms/step - loss: 0.6933 - accuracy: 0.4970 - val_loss: 0.6927 - val_accuracy: 0.5166

Epoch 00001: val_accuracy improved from -inf to 0.51660, saving model to /content/drive/MyDrive/nlpdata/imdb/weights_rnn.h5
Epoch 2/5
157/157 [==============================] - 16s 103ms/step - loss: 0.6928 - accuracy: 0.5002 - val_loss: 0.6919 - val_accuracy: 0.5172

Epoch 00002: val_accuracy improved from 0.51660 to 0.51720, saving model to /content/drive/MyDrive/nlpdata/imdb/weights_rnn.h5
Epoch 3/5
157/157 [==============================] - 16s 103ms/step - loss: 0.6527 - accuracy: 0.5632 - val_loss: 0.3367 - val_accuracy: 0.8568

Epoch 00003: val_accuracy improved from 0.51720 to 0.85680, saving model to /content/drive/MyDrive/nlpdata/imdb/weights_rnn.h5
Epoch 4/5
157/157 [==============================] - 16s 103ms/step - loss: 0.2898 - accuracy: 0.8881 - val_loss: 0.2942 - val_accuracy: 0.8836
```

IMDB 리뷰 감성 분류 - CNN 모델



https://yngie-c.github.io/nlp/2020/07/23/nlp_doc_classification_cnn/

IMDB 리뷰 감성 분류 - CNN 모델



https://arxiv.org/pdf/1510.03820.pdf

IMDB 리뷰 감성 분류 - CNN 모델

데이터 로드

```
[4]  train_input = np.load(open(DATA_PATH + TRAIN_INPUT_DATA, 'rb'))
     train_label = np.load(open(DATA_PATH + TRAIN_LABEL_DATA, 'rb'))
     prepro_configs = prepro_configs = json.load(open(DATA_PATH + DATA_CONFIGS, 'r'))
```

모델 하이퍼파라메터 정의

```
[5]  model_name = 'cnn_classifier'
     BATCH_SIZE = 512
     NUM_EPOCHS = 2
     VALID_SPLIT = 0.2
     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}
```

IMDB 리뷰 감성 분류 - CNN 모델

```
[6]  class CNNClassifier(tf.keras.Model):
         def __init__(self, **kargs):
             super(CNNClassifier, self).__init__(name=kargs['model_name'])
             self.embedding = layers.Embedding(input_dim=kargs['vocab_size'],
                                               output_dim=kargs['embedding_size'])
             self.conv_list = [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 = layers.GlobalMaxPooling1D()
             self.dropout = layers.Dropout(kargs['dropout_rate'])
             self.fc1 = layers.Dense(units=kargs['hidden_dimension'],
                                     activation=tf.keras.activations.relu,
                                     kernel_constraint=tf.keras.constraints.MaxNorm(max_value=3.))
             self.fc2 = 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
```

IMDB 리뷰 감성 분류 - CNN 모델

모델 컴파일

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

Callback 선언

```
[8]  earlystop_callback = EarlyStopping(monitor='val_accuracy', min_delta=0.0001,patience=2)
     checkpoint_path = DATA_PATH + '/weights_cnn.h5'
     cp_callback = ModelCheckpoint(checkpoint_path, monitor='val_accuracy', verbose=1, save_best_only=True, save_weights_only=True)
```

모델 학습

```
[9]  history = model.fit(train_input, train_label, batch_size=BATCH_SIZE, epochs=NUM_EPOCHS,
                         validation_split=VALID_SPLIT, callbacks=[earlystop_callback, cp_callback])
```

```
Epoch 1/2
40/40 [==============================] - 40s 171ms/step - loss: 0.6921 - accuracy: 0.5323 - val_loss: 0.6262 - val_accuracy: 0.7520

Epoch 00001: val_accuracy improved from -inf to 0.75200, saving model to /content/drive/MyDrive/nlpdata/imdb/weights_cnn.h5
```

영문 텍스트 분류 실습

imdb_preprocessing.ipynb

imdb_classification_lr_tfidf.ipynb

imdb_classification_lr_word2vec.ipynb

imdb_classification_rnn.ipynb

imdb_classification_cnn.ipynb

한글 텍스트 분류 실습

nsmc_preprocessing.ipynb

nsmc_classification_cnn.ipynb

nsmc_classification_rnn.ipynb

참고자료

http://www.yes24.com/Product/Goods/92716461

https://github.com/NLP-kr/tensorflow-ml-nlp-tf2

Thank you

kgpark88@gmail.com