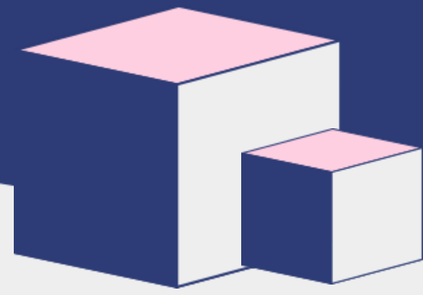
자연어처리 프로젝트

딥러닝 기반 감성분석 및 문서요약 서비스 개발

Bidrectional LSTM 모듈 활용

TeamGPT

김영준 이진균 권영규



목차

개발 배경

서비스 아키텍처 소개

감성분석 아키텍처

문서요약 아키텍처

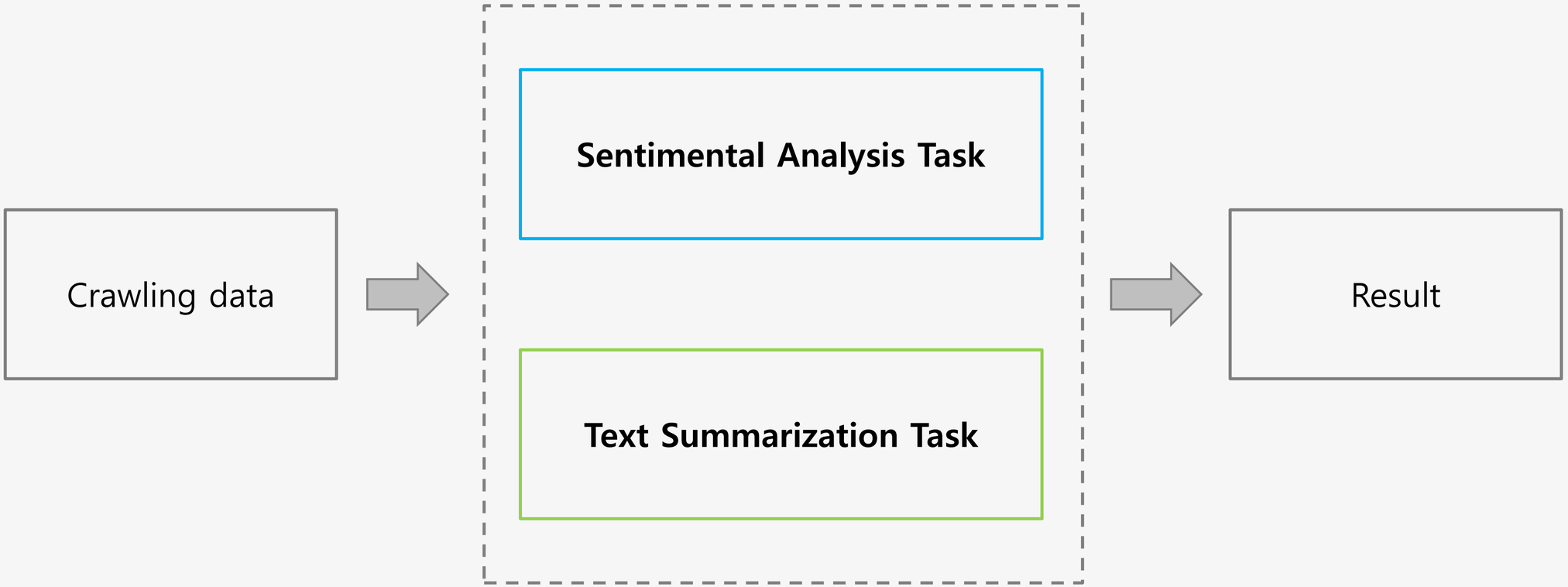
결론

개발 배경

바쁜 일상 속 뉴스를 볼 때

서비스 아키텍처 소개

Core Service



감성분석 아키텍처

데이터 수집

데이터셋

dataset_prep.csv 0.0KB

https://heytech.tistory.com/394

https://github.com/bab2min/corpus

<감성분석 데이터셋 v1>

stopwords-ko.txt 6.1KB

```
with open('./stopwords-ko.txt', 'r') as s:
    stopwords=  s.readlines()

stopwords = [words.strip() for words in stopwords ]
```

참고링크

https://heytech.tistory.com/394
https://github.com/bab2min/corpus

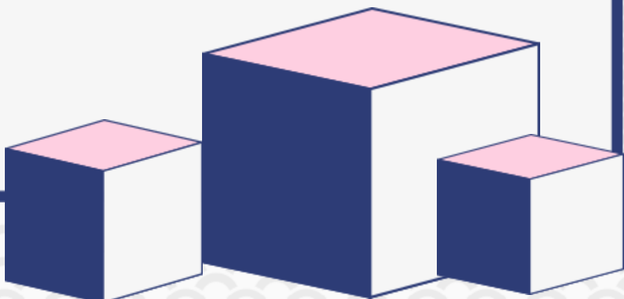
Sources

네이버 영화

네이버 쇼핑

스팀코리아

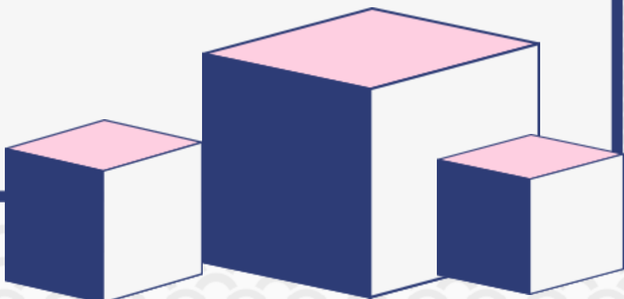
Pharsebank



임포트트

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
import urllib.request
from tqdm import tqdm
import pickle

# from konlpy.tag import Okt
from eunjeon import Mecab

# !pip install tensorflow-addons
import tensorflow_addons as tfa

import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Embedding, Dense, GRU, LSTM , Bidirectional
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import load_model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

from collections import Counter
```

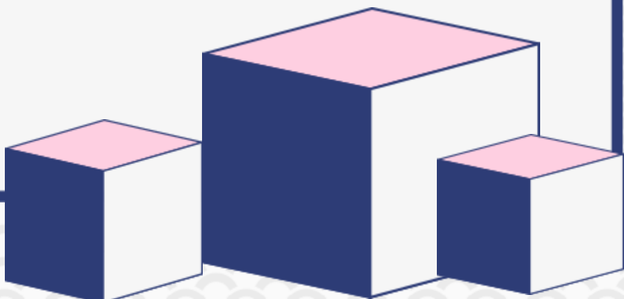
✓ 64s



전처리

```
df_train.document=  df_train.document.str.replace('[^가-힣]' , ' ')
df_valid.document=  df_valid.document.str.replace('[^가-힣]' , ' ')

df_train.document = df_train.document.str.replace('^ +', '')
df_valid.document = df_valid.document.str.replace('^ +', '')

df_train.document = df_train.document.replace('', np.nan)
df_valid.document = df_valid.document.replace('', np.nan)
df_train  =  df_train.dropna(how='any')
df_valid  =  df_valid.dropna(how='any')
df_train = df_train.drop_duplicates()
df_valid = df_valid.drop_duplicates()
df_train.label = df_train.label.apply(lambda x: 1 if x=='pos' else 0 )
df_valid.label = df_valid.label.apply(lambda x: 1 if x=='pos' else 0 )
print(df_train.shape , df_valid.shape)
```

- 한글을 제외한 모든 글자 제거
- 공백 2개 이상 있으면 제거
- 내용이 없으면 공백 처리
- 결측치 제거
- 중복 제거
- 데이터 0과 1로 변환



stopwards 데이터

```
with open('./data/stopwords-ko.txt', 'r') as s:
    stopwords=  s.readlines()

stopwords = [words.strip() for words in stopwords]
```

✓ 0.0s

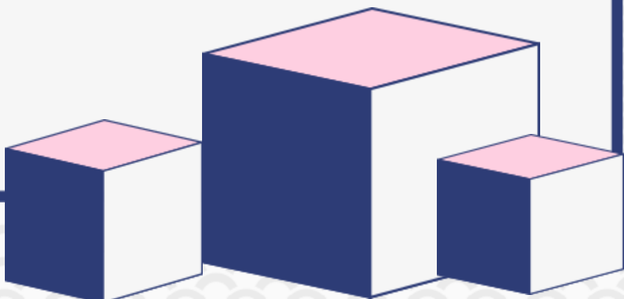
=

stopwords-ko copy.txt | stopwor

data > stopwords-ko.txt

```
1   가
2   가까스로
3   가령
4   각
5   각각
6   각자
7   각종
8   갖고말하자면
9   같다
10  같이
11  개의치않고
12  거니와
13  거바
14  거의
15  것
16  것과 같이
17  것들
18  게다가
```

- 불용어 처리 데이터를 불러옴
- 이후 불용어 전처리 수정

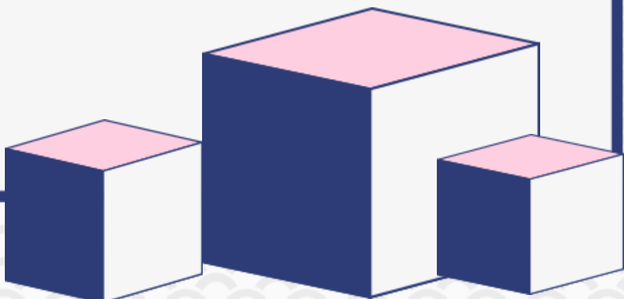


Mecab()

```
mecab = Mecab()


df_train['tokenized'] = df_train['document'].apply(mecab.morphs)
df_train['tokenized'] = df_train['tokenized'].apply(lambda x: [item for item in x if item not in stopwords])
train_data = df_train.copy()



df_valid['tokenized'] = df_valid['document'].apply(mecab.morphs)
df_valid['tokenized'] = df_valid['tokenized'].apply(lambda x: [item for item in x if item not in stopwords])
test_data = df_valid.copy()



negative_words = np.hstack(train_data[train_data.label == 0]['tokenized'].values)
positive_words = np.hstack(train_data[train_data.label == 1]['tokenized'].values)



negative_word_count = Counter(negative_words)
print(negative_word_count.most_common(20))

positive_word_count = Counter(positive_words)
print(positive_word_count.most_common(20))
```

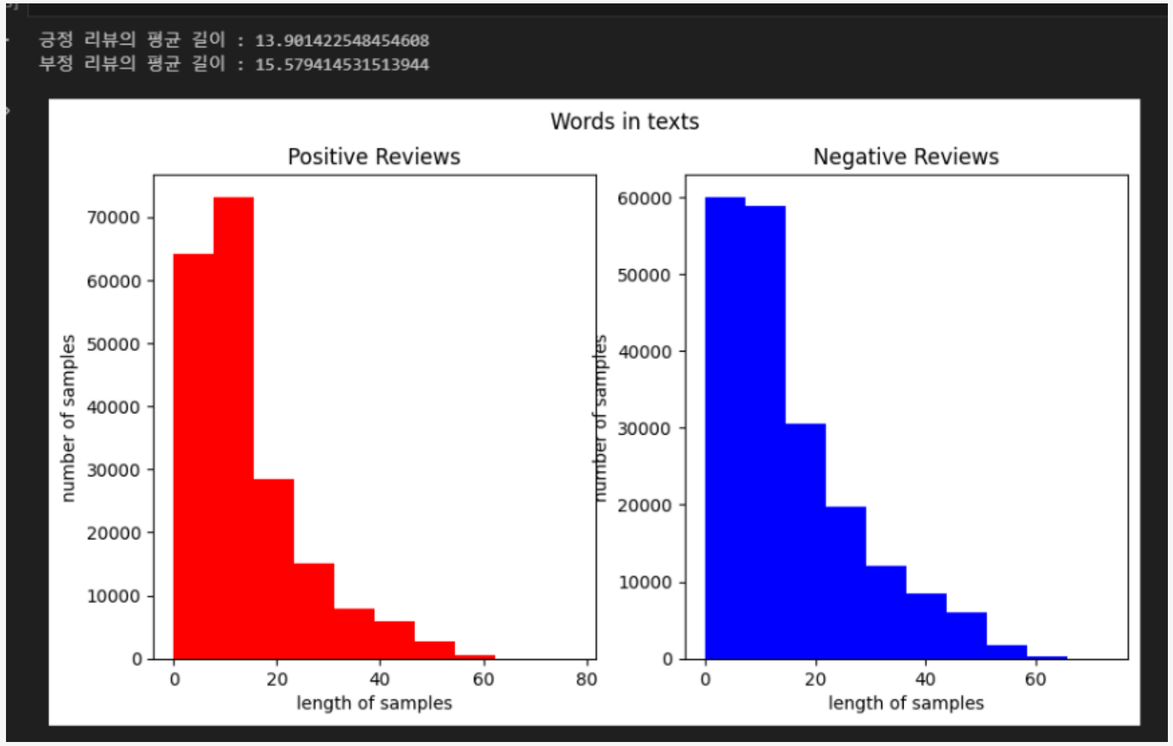
```
, 51924), ('네요', 39948), ('안', 37054), ('는데', 33429), ('지', 31439), ('한', 30762), ('영화', 30282), ('게', 28953), ('없', 28664), ('있',
, 46555), ('은', 41553), ('있', 35226), ('게', 34905), ('영화', 31800), ('네요', 31193), ('한', 29747), ('잘', 27080), ('어요', 26012), ('아요'
```

- mecab.morphs (형태소 분석) 사용
- 두 개의 리스트를 사용하여 단어들의 빈도를 계산
- 가장 빈번하게 나타나는 상위 20개 단어를 출력



긍정 또는 부정 데이터 시각화

```
긍정 리뷰의 평균 길이 : 13.901422548454608
부정 리뷰의 평균 길이 : 15.57941453513944
```



- 긍정 == 1
- 부정 == 0

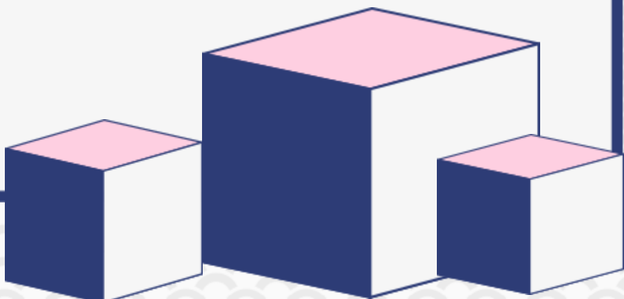
- 긍정 리뷰 평균길이 13.09
- 부정 리뷰 평균 길이 15.57
- matplotlib.pyplot 임포트하여 시각화

Tokenizer()

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)

threshold = 2
total_cnt = len(tokenizer.word_index) # 단어의 수
rare_cnt = 0 # 등장 빈도수가 threshold보다 작은 단어의 개수를 카운트
total_freq = 0 # 훈련 데이터의 전체 단어 빈도수 총 합
rare_freq = 0 # 등장 빈도수가 threshold보다 작은 단어의 등장 빈도수의 총 합

# 단어와 빈도수의 쌍(pair)을 key와 value로 받는다.
for key, value in tokenizer.word_counts.items():
    total_freq = total_freq + value

    # 단어의 등장 빈도수가 threshold보다 작으면
    if(value < threshold):
        rare_cnt = rare_cnt + 1
        rare_freq = rare_freq + value

print('단어 집합(vocabulary)의 크기 :',total_cnt)
print('등장 빈도가 %s번 이하인 희귀 단어의 수: %s'%(threshold - 1, rare_cnt))
print("단어 집합에서 희귀 단어의 비율:", (rare_cnt / total_cnt)*100)
print("전체 등장 빈도에서 희귀 단어 등장 빈도 비율:", (rare_freq / total_freq)*100)


# 전체 단어 개수 중 빈도수 2이하인 단어 개수는 제거.
# 0번 패딩 토큰과 1번 OOV 토큰을 고려하여 +2
vocab_size = total_cnt - rare_cnt + 2
print('단어 집합의 크기 :',vocab_size)
```

```
4] ✓ 2.3s
단어 집합(vocabulary)의 크기 : 73062
등장 빈도가 1번 이하인 희귀 단어의 수: 27125
단어 집합에서 희귀 단어의 비율: 37.12600257315704 5
전체 등장 빈도에서 희귀 단어 등장 빈도 비율: 0.46524280357285897
단어 집합의 크기 : 45939
```

- tokenizer 함수로 텍스트 데이터 토큰화
- threshold : 희귀 단어 판단을 위한 빈도수 임계값
- total_cnt : x_train 안에 있는 총 단어 개수
- rare_cnt : 등장 빈도수가 'threshold'보다 작은 희귀 단어의 개수
- total_freq : x_train 의 전체 단어 빈도수의 총 합
- rare_freq : 등장 빈도수가 'threshold'보다 작은 희귀 단어의 등장 빈도수의총 합
- vocab_size : 희귀 단어를 제거한 뒤의 단어 집합의 크기



Tokenizer()

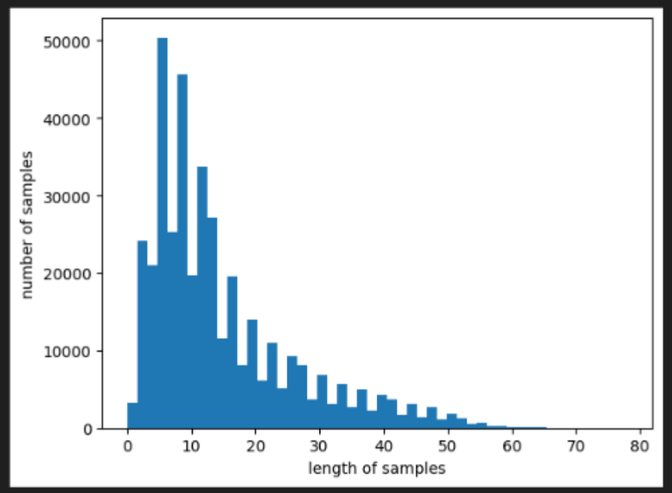
```
tokenizer = Tokenizer(vocab_size, oov_token = 'OOV')
tokenizer.fit_on_texts(X_train)
X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)
```

[16] ✓ 5.2s

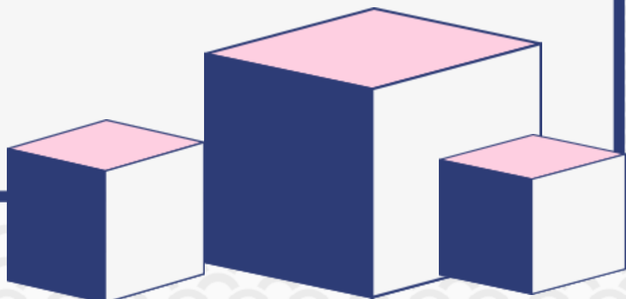
```
print('리뷰의 최대 길이 :',max(len(review) for review in X_train))
print('리뷰의 평균 길이 :',sum(map(len, X_train))/len(X_train))
plt.hist([len(review) for review in X_train], bins=50)
plt.xlabel('length of samples')
plt.ylabel('number of samples')
plt.show()
```

[17] ✓ 0.7s

```
리뷰의 최대 길이 : 78
리뷰의 평균 길이 : 14.740223393066623
```



- tokenizer = Tokenizer(vocab_size, oov_token='OOV'):
- Tokenizer 객체를 생성
- vocab_size : 어휘 사전(vocabulary)에 포함시킬 최대 단어 수를 결정함
- oov_token : 학습 과정에서 본 적 없는 단어(Out-of-Vocabulary, OOV)를 대체하기 위한 토큰을 지정

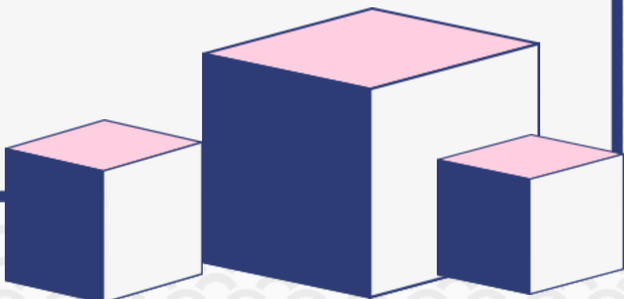


Naive Bayes Classifier

```
mod = MultinomialNB()
mod.fit(x_train, y_train)
```

- 텍스트 분류를 위해 전통적으로 사용되는 분류기로 나이브 베이즈 분류기를 사용했습니다. 나이브 베이즈 분류기는 인공 신경망 알고리즘에는 속하지 않지만, 머신 러닝의 주요 알고리즘으로 분류에 있어 준수한 성능을 보여주는 것으로 알려져 있습니다.
- 나이브 베이즈 분류기에서 토큰화 이전의 단어의 순서는 중요하지 않습니다. BoW와 같이 단어의 순서를 무시하고 오직 빈도수만을 고려한다는 단점이 있습니다.

약 73%의 Accuracy가 나왔습니다.



CNN Classifier

```python
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['embbeding_size'])
    self.conv_list = [layers.Conv1D(filters=kargs['num_filters'], kernel_size=kernel_size, padding='valid',activation ='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 = 'relu',
                            kernel_constraint=tf.keras.constraints.MaxNorm(max_value=3.))
    self.fc2 = layers.Dense(units=1 ,
                            activation='sigmoid',
                            kernel_constraint= tf.keras.constraints.MaxNorm(max_value=3.))

  def call(self,x):
    embedded = self.embedding(x)
    dropouted = self.dropout(embedded)
    concat_conv = tf.concat([self.pooling(conv(dropouted)) for conv in self.conv_list], axis = 1)
    fc = self.fc1(concat_conv)
    output = self.fc2(fc)
    return output

# from tensorflow.keras.models import save_model

model = CNNClassifier(**kargs)
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0002),
              loss = 'binary_crossentropy',
              metrics = ['acc'])
```

- 임베딩 이후 Conv1D 를 사용해서 합성곱을 해주고 GlobalMaxPooling1D()은 1D 시퀀스 데이터에 대해 전역 최대 풀링(global max pooling)을 수행하는 Keras 레이어입니다.
- 이후 Fully Connected Dense레이어를 거친 이후 시그모이드로 나오게 해줬습니다.

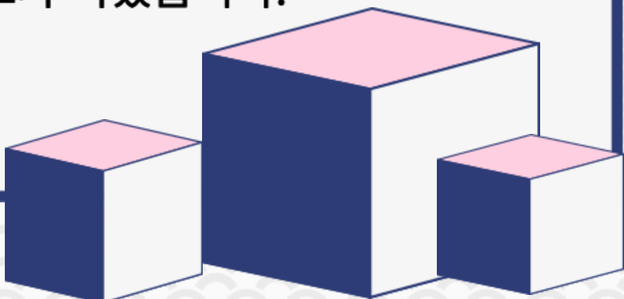
장점

- 지역적 특징 감지
- 파라미터 공유

단점

- 긴 문장 처리
- 전역적 의미 파악 어려움
- 단어의 순서 무시

약 75%의 정확도가 나왔습니다.

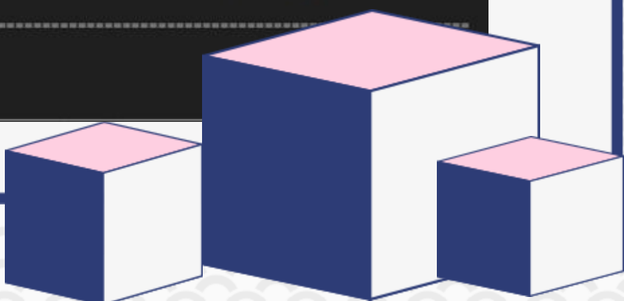


다국적 Bert

```
bert_preprocess = hub.KerasLayer('https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3')
bert_encoder = hub.KerasLayer('https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/4')


text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
preprocessed_text = bert_preprocess(text_input)
outputs = bert_encoder(preprocessed_text)
l = tf.keras.layers.Dropout(0.1, name='dropout')(outputs['pooled_output'])
l = tf.keras.layers.Dense(1, activation='sigmoid', name= 'output')(l)

model = tf.keras.Model(inputs= [text_input], outputs=[l])


model.summary()
✓ 21.9s
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
text (InputLayer)               [(None,)]            0           []

keras_layer (KerasLayer)        {'input_mask': (Non  0           ['text[0][0]']
                                e, 128),
                                 'input_word_ids':
                                (None, 128),
                                 'input_type_ids':
                                (None, 128)}

keras_layer_1 (KerasLayer)      {'pooled_output': (  109482241   ['keras_layer[0][0]',
                                None, 768),                       'keras_layer[0][1]',
                                 'sequence_output':               'keras_layer[0][2]']
                                (None, 128, 768),
                                'encoder_outputs':
                                [(None, 128, 768),
                                (None, 128, 768),
                                (None, 128, 768),
                                (None, 128, 768),
                                (None, 128, 768),
                                (None, 128, 768),
                                (None, 128, 768),
                                (None, 128, 768),
                                (None, 128, 768),
                                (None, 128, 768),
                                (None, 128, 768),
                                (None, 128, 768)],
                                'default': (None,
                                768)}

dropout (Dropout)               (None, 768)          0           ['keras_layer_1[0][13]']

output (Dense)                  (None, 1)            769         ['dropout[0][0]']

==================================================================================================
Total params: 109,483,010
Trainable params: 769
Non-trainable params: 109,482,241
__________________________________________________________________________________________________
```

- Hugging Face 에서 다국적 BERT 모델을 다운 받고
공식 문서에서 확인 결과 Second dimension이 128 이고 features dimension이 768으로 맞춰줘야합니다.
- 이걸 자동으로 해주는 bert preprocess와 bert encoder를 사용해서 전처리를 해주고 시그모이드로 나오게 해줬습니다.
- 파라미터수가 1억개이며 1 epoch에 1시간이 넘는 학습시간과 accuracy의 상승폭이 84% 이후 너무 작아 다른 모델을 사용하기로 결정했습니다.



GRU

```python
from tensorflow.keras.layers import Embedding, Dense, GRU
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import load_model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

embedding_dim = 100
hidden_units = 128

# tfa.metrics.F1Score()

model = Sequential()
model.add(Embedding(vocab_size, embedding_dim))
model.add(GRU(hidden_units))
model.add(Dense(1, activation='sigmoid'))


precision = tf.keras.metrics.Precision(name='precision')
recall = tf.keras.metrics.Recall(name='recall')

# es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
mc = ModelCheckpoint('best_model.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc',precision,recall, tfa.metrics.F1Score(name='f1_score', num_classes=1)])
history = model.fit(X_train, y_train, epochs=50, callbacks=[mc], batch_size=64, validation_split=0.2)
```

1m 15.9s Python

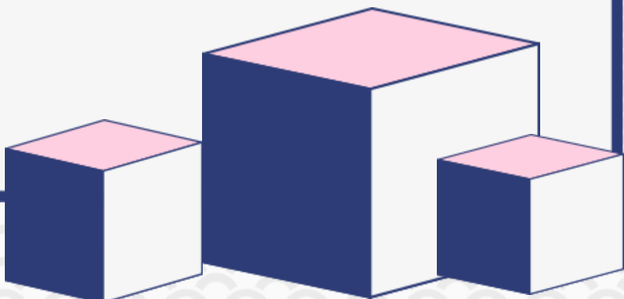
```
Epoch 1/50
4943/4945 [============================>.] - ETA: 0s - loss: 0.3224 - acc: 0.8640 - precision: 0.8700 - recall: 0.8560 - f1_score: 0.6667
Epoch 1: val_acc improved from -inf to 0.73654, saving model to best_model.h5
4945/4945 [==============================] - 54s 10ms/step - loss: 0.3224 - acc: 0.8640 - precision: 0.8700 - recall: 0.8560 - f1_score: 0.6667 - val_loss: 0.5414 - val_acc: 0.7365 - val_precision: 0.8159 - val_r
Epoch 2/50
1729/4945 [=========>....................] - ETA: 27s - loss: 0.2764 - acc: 0.8880 - precision: 0.8913 - recall: 0.8834 - f1_score: 0.6660
```

- 임베딩 이후 GRU layer 사용 이후, 시그모이드로 나오게 했습니다.

장점
- LSTM 보다 더 간단한 구조
- 경량화

단점
- 기억 능력

약 85%의 정확도가 나왔습니다.



LSTM

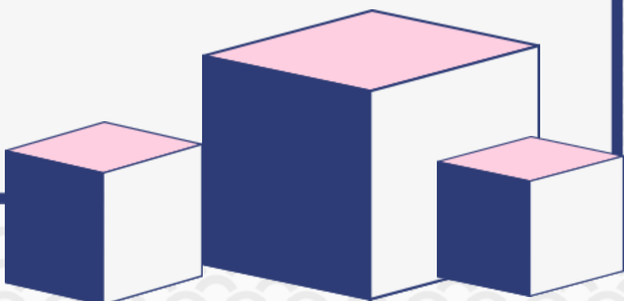
```python
from tensorflow.keras.layers import Embedding, Dense, GRU
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import load_model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

embedding_dim = 100
hidden_units = 128

# tfa.metrics.F1Score()

model = Sequential()
model.add(Embedding(vocab_size, embedding_dim))
model.add(LSTM(hidden_units))
model.add(Dense(1, activation='sigmoid'))


precision = tf.keras.metrics.Precision(name='precision')
recall = tf.keras.metrics.Recall(name='recall')

# es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
mc = ModelCheckpoint('best_model.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc',precision,recall, tfa.metrics.F1Score(name='f1_score', num_classes=1)])
history = model.fit(X_train, y_train, epochs=50, callbacks=[mc], batch_size=1000, validation_data=[X_test, y_test])
```

```
Epoch 1/50
395/396 [============================>.] - ETA: 0s - loss: 0.4118 - acc: 0.8145 - precision: 0.8124 - recall: 0.8180 - f1_score: 0.6667
Epoch 1: val_acc improved from -inf to 0.84422, saving model to best_model.h5
396/396 [==============================] - 7s 15ms/step - loss: 0.4117 - acc: 0.8146 - precision: 0.8125 - recall: 0.8180 - f1_score: 0.6668 - val_loss: 0.3613 - val_acc: 0.8442 -
Epoch 2/50
395/396 [============================>.] - ETA: 0s - loss: 0.3305 - acc: 0.8595 - precision: 0.8639 - recall: 0.8536 - f1_score: 0.6668
Epoch 2: val_acc improved from 0.84422 to 0.85236, saving model to best_model.h5
396/396 [==============================] - 6s 14ms/step - loss: 0.3305 - acc: 0.8595 - precision: 0.8639 - recall: 0.8536 - f1_score: 0.6668 - val_loss: 0.3429 - val_acc: 0.8524 -
Epoch 3/50
393/396 [============================>.] - ETA: 0s - loss: 0.2911 - acc: 0.8785 - precision: 0.8814 - recall: 0.8748 - f1_score: 0.6668
Epoch 3: val_acc improved from 0.85236 to 0.85621, saving model to best_model.h5
396/396 [==============================] - 6s 14ms/step - loss: 0.2911 - acc: 0.8785 - precision: 0.8814 - recall: 0.8748 - f1_score: 0.6668 - val_loss: 0.3373 - val_acc: 0.8562 -
```

장점

- 장기 의존성 학습
- 더 강력한 기억 구조

약 86%의 정확도가 나왔습니다.



Bidirectional LSTM

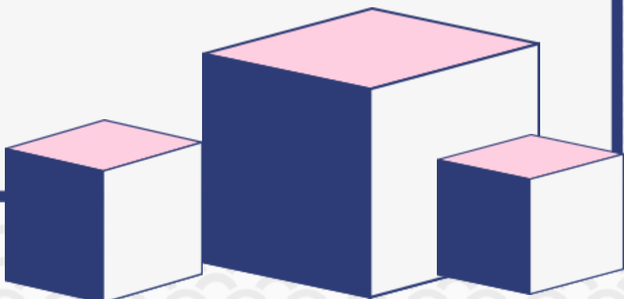
```python
embedding_dim = 128
hidden_units = 256

# tfa.metrics.F1Score()

model = Sequential()
model.add(Embedding(vocab_size, embedding_dim))
model.add(Bidirectional(LSTM(hidden_units)))
model.add(Dense(1, activation='sigmoid'))


precision = tf.keras.metrics.Precision(name='precision')
recall = tf.keras.metrics.Recall(name='recall')

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=15)
mc = ModelCheckpoint('best_model.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)

# model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc',precision,recall, tfa.metrics.F1Score(name='f1_score', num_classes=1)])
model.compile(optimizer='Adam', loss='binary_crossentropy', metrics=[tfa.metrics.F1Score(name='f1_score', num_classes=1), 'acc', precision, recall])
history2 = model.fit(X_train, y_train, epochs=50, callbacks=[checkpointer_acc,checkpointer_loss, mc, es], batch_size=128, validation_data=[X_test, y_test])
```

```
51]
..
=======] - ETA: 0s - loss: 0.3701 - f1_score: 0.6668 - acc: 0.8362 - precision: 0.8398 - recall: 0.8309
: to 0.85885, saving model to ./model_acc4\01-0.85885.hdf5

: to 0.32930, saving model to ./model_loss4\01-0.32930.hdf5

: to 0.85885, saving model to best_model.h5
=======] - 49s 13ms/step - loss: 0.3701 - f1_score: 0.6668 - acc: 0.8362 - precision: 0.8398 - recall: 0.8309 - val_loss: 0.3293 - val_f1_score: 0.6668 - val_acc: 0.8589

====>.] - ETA: 0s - loss: 0.3000 - f1_score: 0.6668 - acc: 0.8730 - precision: 0.8752 - recall: 0.8701
;885 to 0.86081, saving model to ./model_acc4\02-0.86081.hdf5

;2930 to 0.32585, saving model to ./model_loss4\02-0.32585.hdf5

;885 to 0.86081, saving model to best_model.h5
=======] - 43s 14ms/step - loss: 0.3000 - f1_score: 0.6668 - acc: 0.8730 - precision: 0.8752 - recall: 0.8701 - val_loss: 0.3258 - val_f1_score: 0.6668 - val_acc: 0.8608
```

장점
- 양방향 정보 활용
- 문맥 파악

단점
- 계산 비용
- 메모리 사용

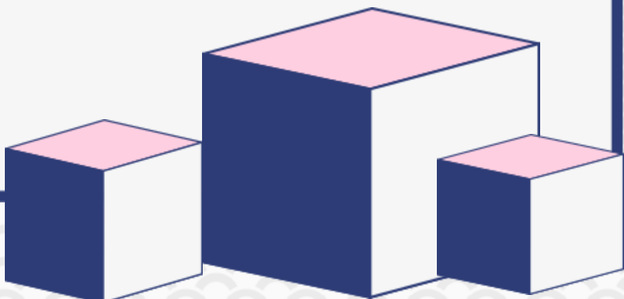
87.11%의 정확도가 나왔습니다.



Crawling 및 예측

```python
from selenium import  webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.chrome.service import Service
# from webdriver_manager.chrome import ChromeDriverManager
import time

def find_movie_reviews(search,  n: int )  ->  list:
    options = Options()

    options.add_experimental_option('detach', True)

    # service = Service(ChromeDriverManager().install())
    options.add_argument('--start-maximized')
    driver = webdriver.Chrome(options=options)
    url =f'https://search.naver.com/search.naver?where=nexearch&sm=top_sug.pat&fbm=0&acr=1&acq=%EC%8A%A4%EC%A6%88%EB%A9%94%EC%9D%98+%EB%AC%B8%EB%8B%A8%EC%86%8D+%ED%8F%89%EC%A0%90&qdt=0&ie=utf8&query={search}'

    driver.get(url)

    driver.implicitly_wait(time_to_wait=10)
    area_card_outer = driver.find_element(By.CSS_SELECTOR, '.area_card_outer._item_wrapper')
    area_card_items = area_card_outer.find_elements(By.CSS_SELECTOR  , '.area_card._item')
    reviews= []
    for area_card_item in area_card_items:

        review  =  area_card_item.find_element(By.CSS_SELECTOR  ,'.desc._text').text
        reviews.append(review)
        n -=1
        if n==0:
            break
    driver.close()
    return reviews

input_from_client = input()
reviews = find_movie_reviews(input_from_client, 10)

for review in reviews:
    print(review)
    print(sentiment_predict(review))
```

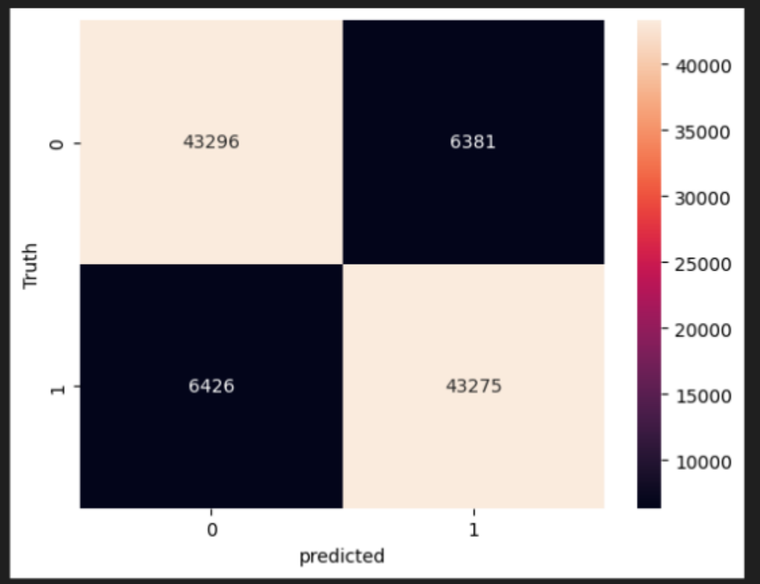
```
경찰 '신림동 칼부림' 30대 얼굴·이름 공개 오늘 결론
1/1 [==============================] - 0s 41ms/step
66.22% 확률로 부정 리뷰입니다.
None
칼부림 전날 PC 부수고 폰 초기화...신림 살해범 "범행 계획했다"
1/1 [==============================] - 0s 41ms/step
67.50% 확률로 긍정 리뷰입니다.
None
10분전 흉기 훔쳐 신림동 도착 직후 칼부림(종합2보)
1/1 [==============================] - 0s 53ms/step
94.21% 확률로 부정 리뷰입니다.
None
10분 전 흉기 훔쳐 신림동 도착 직후 칼부림
1/1 [==============================] - 0s 42ms/step
83.37% 확률로 부정 리뷰입니다.
None
칼부림 피의자, 신림동 난동 이번 처음 아니야
1/1 [==============================] - 0s 54ms/step
73.23% 확률로 부정 리뷰입니다.
None
'묻지마' 흉기난동범 10분전 흉기 훔쳐 신림동 도착 직후 칼부림
1/1 [==============================] - 0s 46ms/step
95.31% 확률로 부정 리뷰입니다.
None
'묻지마 칼부림' 트라우마 겪는 신림동 골목 상인들
...
신림동 칼부림 6일 전_제기동서도 39cm 흉기난동 있었다
1/1 [==============================] - 0s 50ms/step
78.58% 확률로 부정 리뷰입니다.
None
```



평가지표

```
import matplotlib.pyplot  as plt
import seaborn as sns

sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('predicted')
plt.ylabel('Truth')
```

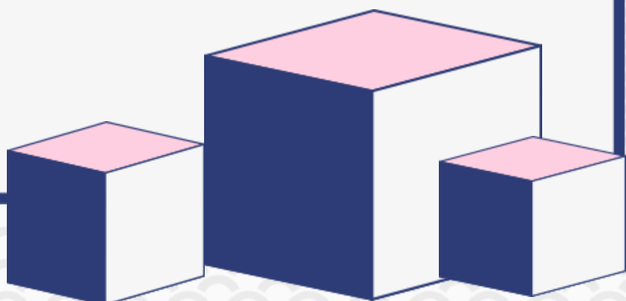
```
Text(50.722222222222214, 0.5, 'Truth')
```




```
print(classification_report(y_test , result))
```

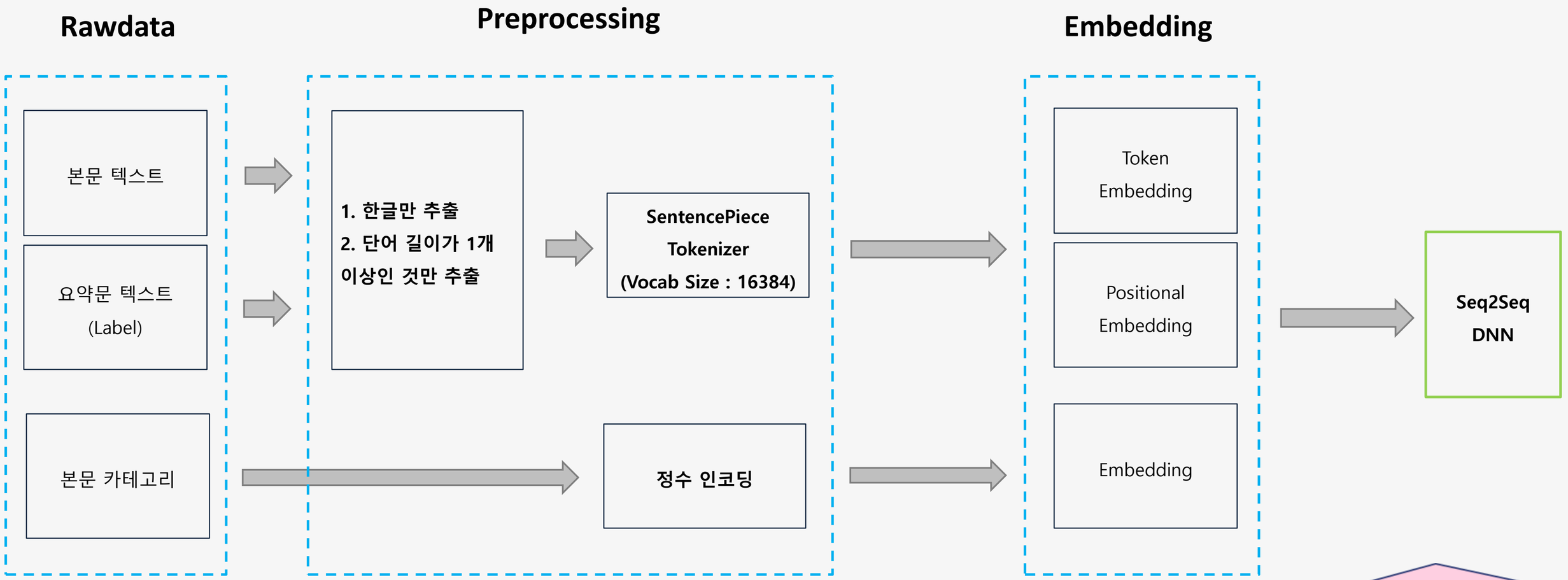
✓ 0.1s

	precision	recall	f1-score	support
0	0.87	0.87	0.87	49677
1	0.87	0.87	0.87	49701
accuracy			0.87	99378
macro avg	0.87	0.87	0.87	99378
weighted avg	0.87	0.87	0.87	99378

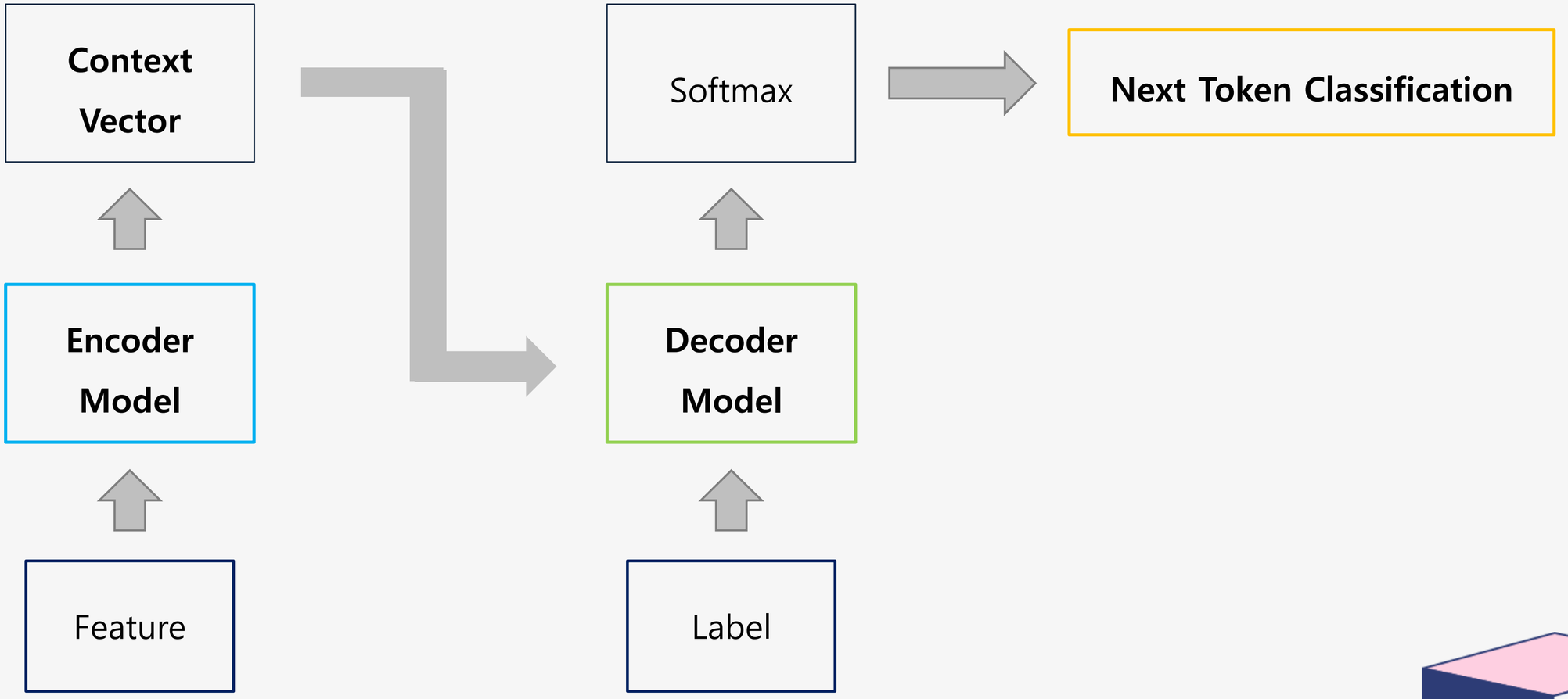


문서요약 아키텍처

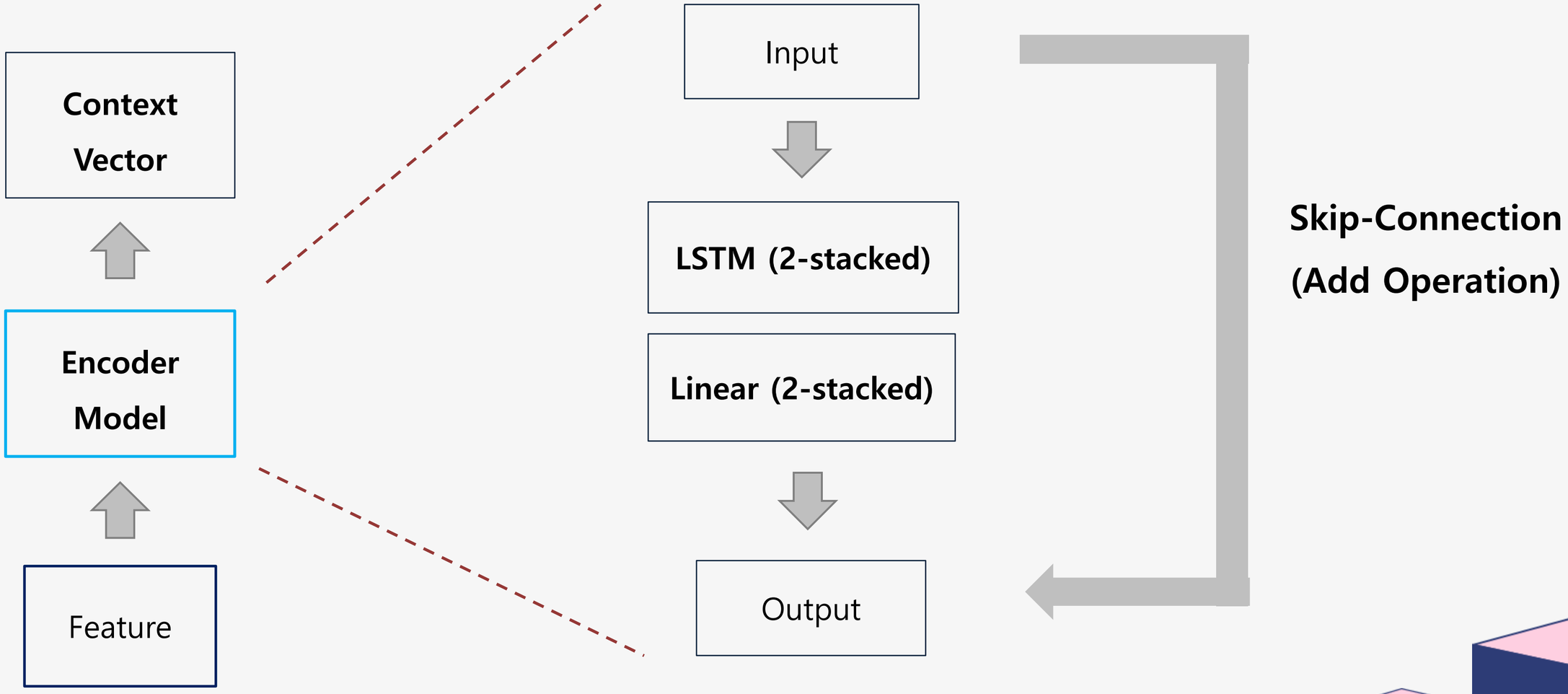
문서요약 아키텍처 - 인풋 구성



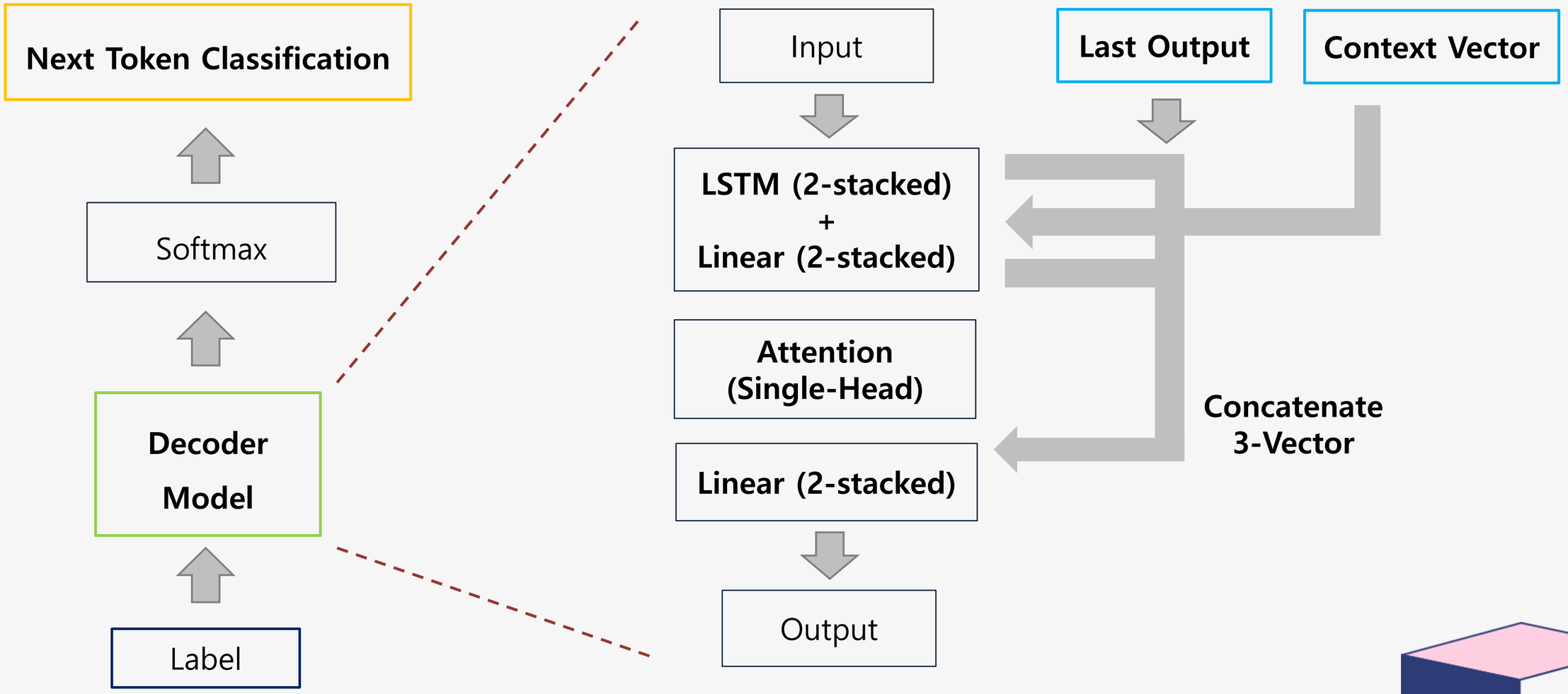
문서요약 아키텍처 - Seq2Seq 모델 아키텍처



문서요약 아키텍처 - 인코더



문서요약 아키텍처 - 디코더



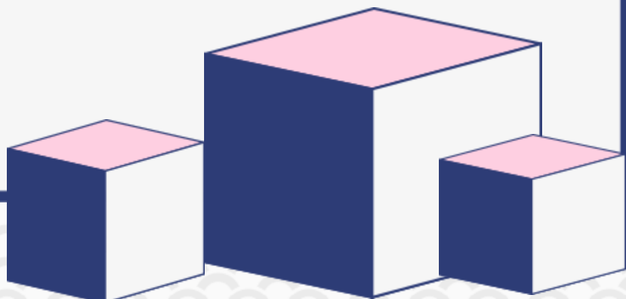
문서요약 아키텍처 - 학습 하이퍼파라미터 및 평가 결과

Training Hyper-Parameters

1. Batch size : 64 (Iteration : Appox. 1000)
2. Epochs : 30
3. Early Stopping Sounds : 10
4. Loss : Cross Entropy
5. Optimizer : AdamW
    * Learning Rate : 5e-4
    * Weight Decay : 1e-4

Evaluation Table

Train Loss	2.40039
Valid Loss	2.57735
Train Accuracy	**71.33%**
Valid Accuracy	**72.64%**



문서요약 아키텍처 - 검증 데이터 추론 및 오프라인 평가

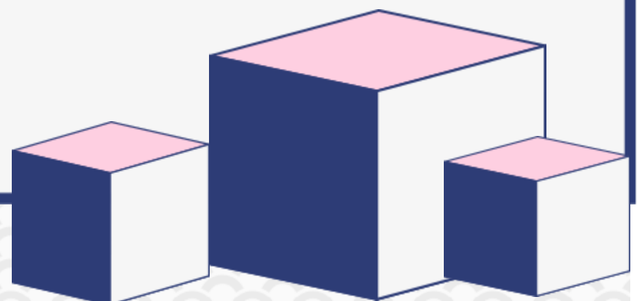
추론 결과 예시 – Best Case

요약된 본문

그다음 수요일 오후에 이러한 OECD 과학기술 장관회의 결과를 모아서 '대전선언문'이라는 형식으로 발표하게 됩니다. 그래서 OECD 전체 과기장관회의에 이어서 OECD 내에 과학기술정책위원회가 있습니다.
세계과학정상회의 마지막 날은 OECD 과학기술 장관회의, 과학기술포럼의 결과를 우리가 집대성을 해서 한 번 더 과학기술을 통해서 한번 제2도약을 해보자고 하는 대한민국 과학발전 대토론회를 하고, 관련되는 우리 과학기술계의 주요한 분들이 다 오셔서 논의를 하게 됩니다.

모델 요약문

그래서 과학기술장관들이 논의한 결과를 토대로 하여튼 협의를 거쳐서 대전선언문을 확정해서 발표하게 됩니다



문서요약 아키텍처 - 검증 데이터 추론 및 오프라인 평가

추론 결과 예시 – Bad Case

요약된 본문

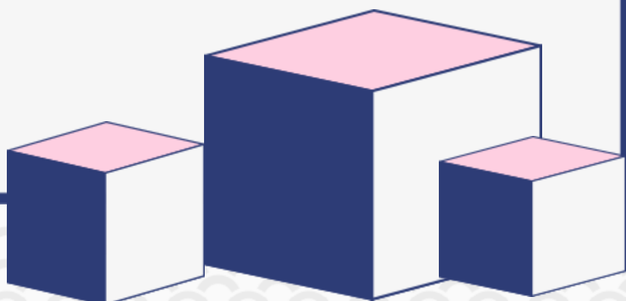
다음으로, 콘텐츠산업의 혁신성장을 주도하는 창의인재 양성 관련입니다. 우선 영화, 게임, 방송영상, 웹툰, 애니메이션 등 분야별 특성을 반영한 맞춤형 교육과정을 운영합니다.

모델 요약문

콘텐츠산업 경쟁력 강화를 위한 일자리 창출을 위해 콘텐츠 기업의 성장단계별 맞춤형 창업 지원을 지속하고 콘텐츠

원인 분석

문장이 어디서 마쳤는지 그에 대한 위치를 전처리 및 토큰화 과정에서 반영하지 않았기 때문인 것으로 추정



문서요약 아키텍처 - 검증 데이터 추론 및 오프라인 평가

추론 결과 예시 – Worst Case

요약된 본문

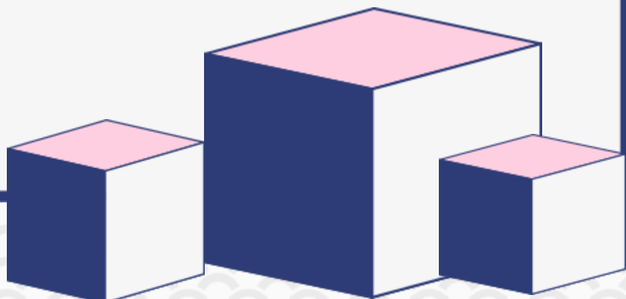
베트남에서는 방향과 관계 없이 무조건 오토바이 앞바퀴를 먼저 내민 사람이 우선권을 갖는다고 한다.
달인 수준의 운전실력도 흔하게 볼 수 있다.

요약된 본문

미국의의은에의의의의은에의의의의의의의

원인 분석

완전한 한글만 추출하여 학습했기에, **숫자 및 기타 정보가 결합되어 있는 문장을 학습하지 못했기 때문으로 추정**
Ex. 코로나19, 3분기GDP, LG전자



결론

향후 발전 방향 - 감성분석

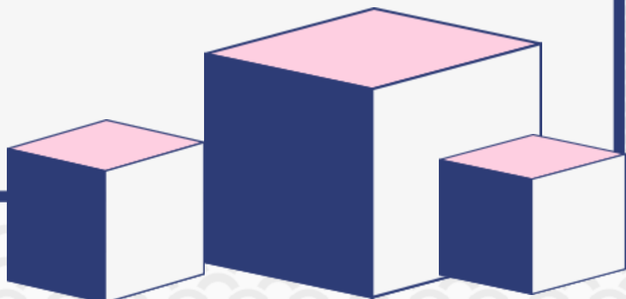
```
df_all.loc[[9,26,50841, 504836, 504825, 504818 , 504803 , 504797 , 23],:]
```

✓ 0.0s

	document	label	src
9	바보가 아니라 병 쉰 인듯	pos	naver
26	사람이 어떻게 저런짓을 할 수가 있는지—— 보는 내가 다 화나더라	pos	naver
50841	암울한 청춘 . 오늘도 신나게 달린다	pos	naver
504836	3 는?	pos	steam
504825	안무서웡	neg	steam
504818	게임하면서 나도 싸이코가됨	pos	steam
504803	하 내3일 어디갔냐 쉬이불.. 성태야 방송켜라	pos	steam
504797	깨지긴하네?	pos	steam
23	강압적용서,세뇌적용서에 대한 비판	pos	naver

- 데이터가 네이버 쇼핑 20만, 네이버 영화 리뷰 20만 , 스팀 리뷰 10만 , 금융뉴스 관련 헤드라인 2천개
- 각각의 데이터만을 가지고 모델을 학습해서 돌렸을시 90프로가 넘는 성능을 보였다.
- 그러나 데이터를 합치다보니 분야가 달라져서 성능이 낮아졌다는 것을 예측을 할 수 있었다.

- 모델을 더욱 발전시키기 위해서는 3가지의 결론을 내릴 수 있었다.
- 더 많은 데이터와 라벨이 잘 되어있는 데이터
- Positive, negative 뿐만아니라 Neutral 까지 있으면 긍정 부정으로 결정하기 어려운 경우 Neutral로 하면 더 모델의 성능이 더 좋아 질 수 있다.



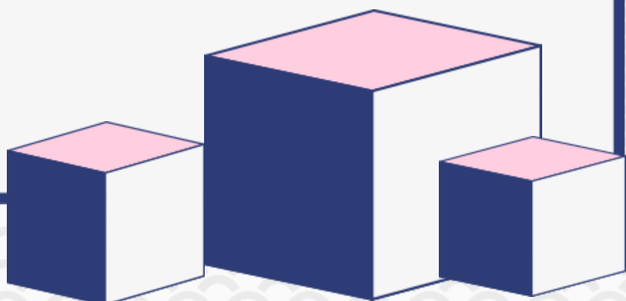
향후 발전 방향 - 문서요약

PreTrained Model 사용 (BERT, RoBERTa, MPNet 등)

사전 학습된 모델을 사용하여 Few-Shot Learning을 통해 고효율의 서비스를 구현

ChatGPT, Bard 등 LLM 에 대한 프롬프트 엔지니어링

프롬프트 엔지니어링을 통해 요약, 감성분석, 엔티티 추출 등 다양한 task를 요청하여 서비스를 구현



감사합니다