인공지능 고급 언어 프로젝트 개발

1주차:자연어처리 개요

- 자연어를 분석하여 인공어로 처리하는 기술
- 인간과 컴퓨터 사이의 통역사 같은 역할

2주차:Word Embedding

I always uh do the main um processing, I mean, the uh um data-processing.

-> I always do the main processing, the data processing. (의미없는 단어들 제거)

구문 중에 의미 없는 단어들을 제거한다.

```
# 토스마스터즈 : 영어 발표 관련한 모임
# 단어를 처리하기 전에, 그럼 어떤 것이 단어인가?
# 단어의 정의가 필요하고...
```

[ ] Python

```
# Vocabulary
# 유사한 단어 형태들의 집합
```

[ ] Python

I always uh do the main um processing, I mean, the uh um data-processing.

-> { I, always, uh, do, the, main, um, processing, mean, data } (Vocabulary)

말뭉치 (Corpus)

구문을 단어별로 나눠서 저장한다.

```
# Tokens : 문서 및 문장에서 단어들 (Tokens)의 리스트 (목록)
# I always uh do the main um processing, I mean, the uh um data-processing.
# 1   2     3  4  5   6   7     8        9  10   11  12 13  14      15
```

[1] Python

```
import spacy
import re
```

[1] Python

```
# python -m spacy download en_core_web_sm
nlp = spacy.load("en_core_web_sm")
```

[2] Python

```
doc = "I always uh do the main um processing, I mean, the uh um data-processing."
```

[6] Python

2주차:Word Embedding

```
stats = nlp(doc)
type(doc), doc, type(stats), stats
```
[18]

```
(str,
 'I always uh do the main um processing, I mean, the uh um data-processing.',
 spacy.tokens.doc.Doc,
 I always uh do the main um processing, I mean, the uh um data-processing.)
```

```
for token in stats:
    print(token.text, type(token.text), token, type(token))
```
[16]

```
I <class 'str'> I <class 'spacy.tokens.token.Token'>
always <class 'str'> always <class 'spacy.tokens.token.Token'>
uh <class 'str'> uh <class 'spacy.tokens.token.Token'>
do <class 'str'> do <class 'spacy.tokens.token.Token'>
the <class 'str'> the <class 'spacy.tokens.token.Token'>
main <class 'str'> main <class 'spacy.tokens.token.Token'>
um <class 'str'> um <class 'spacy.tokens.token.Token'>
processing <class 'str'> processing <class 'spacy.tokens.token.Token'>
, <class 'str'> , <class 'spacy.tokens.token.Token'>
I <class 'str'> I <class 'spacy.tokens.token.Token'>
mean <class 'str'> mean <class 'spacy.tokens.token.Token'>
, <class 'str'> , <class 'spacy.tokens.token.Token'>
the <class 'str'> the <class 'spacy.tokens.token.Token'>
uh <class 'str'> uh <class 'spacy.tokens.token.Token'>
um <class 'str'> um <class 'spacy.tokens.token.Token'>
data <class 'str'> data <class 'spacy.tokens.token.Token'>
- <class 'str'> - <class 'spacy.tokens.token.Token'>
processing <class 'str'> processing <class 'spacy.tokens.token.Token'>
. <class 'str'> . <class 'spacy.tokens.token.Token'>
```

```
doc2 = "Korea has a reasonable population." #구문을 단어 단위로 나눈다.
stats = nlp(doc2)
for token in stats:
    print(token.text)
```
[17] Python

```
Korea
has
a
reasonable
population
.
```

구문을 단어 단위로 나눈다.

```
# spacy의 nlp 방식 말고 이전에 re 방식으로 doc를 token화 해서 for로 출력해보세요!
doc = "I always uh do the main um processing, I mean, the uh um data-processing."

for token in re.split("\W+", doc):
    print(token, type(token))
```
[ ] Python

```
L = list(nlp.vocab.strings)
type(nlp.vocab.strings), type(L), len(L)
```
[24] Python

```
(spacy.strings.StringStore, list, 83814)
```

```
L[50000], L[50001], L[83813]  # en_core_web_sm에서 특정 위치의 단어 보기
```
[28] Python

```
('glands', 'glandular', '¥dd')
```

```
!python -m spacy download en_core_web_trf
```
[29] Python

```
^C
```

```
# python -m spacy download en_core_web_trf
nlp_accuracy = spacy.load("en_core_web_trf")
#L_accuracy = list(nlp_accuracy.vocab.strings)
#type(nlp_accuracy.vocab.strings), type(L_accuracy), len(L_accuracy)
```
[ ] Python

2주차:Word Embedding

```
import numpy as np
import pandas as pd
from collections import Counter
import re
```
[3] Python

```
df = pd.read_csv("reviews.csv")
```
[4] Python

```
df.head(20) #구문을 분석하여 부정적인 단어와 긍정적인 단어를 분류한다.
```
[5] Python

	rating	review
0	negative	terrible place to work for i just heard a stor...
1	negative	hours , minutes total time for an extremely s...
2	negative	my less than stellar review is for service . w...
3	negative	i m granting one star because there s no way t...
4	negative	the food here is mediocre at best . i went aft...
5	negative	n n nwe looked at our entertainment book for ...
6	negative	i had an appointment that was made months in a...
7	negative	i frankly don t get why people like this place...
8	negative	thought i would give them a nd chance . . . ba...
9	negative	if i could give it a negative star i would ! f...
10	negative	i agree , i had a similar situation when i wen...
11	negative	petsmart is the worst place to take your dog ....
12	negative	a pain to get in and out of . i dread each tri...
13	negative	really great staff friendly and professional ....
14	negative	if i could give this place less than one star ...
15	negative	my husband and i ate here after i purchased a ...
16	negative	lobby is very nice . service was nice but weir...
17	negative	i ve been mis diagnosed many times by dr . aus...
18	negative	this use to be one of my favorite ulta locatio...
19	negative	westin is a chain , and thus you have expectat...

구문 중에서 부정적인 의견을 추출하였다.

```
df.tail()
```
[6] Python

	rating	review
55995	positive	great food . wonderful , friendly service . i ...
55996	positive	charlotte should be the new standard for moder...
55997	positive	get the encore sandwich ! ! make sure to get i...
55998	positive	i m a pretty big ice cream gelato fan . pretty...
55999	positive	where else can you find all the parts and piec...

구문들 중 긍정적인 의견을 추출하였다.

3주차: Classification

```
[2]
import spacy
nlp = spacy.load("en_core_web_md")
```
Python

```
[3]
doc = nlp("Alicia and me went to the school by bus")
```
Python

```
[9]
# pos : part of speech

for token in doc:
    print(token.text, "\t", token.pos_, "\t", token.tag_, "\t", spacy.explain(token.pos_), "\t\t", spacy.explain(token.tag_))
```
Python

```
Alicia   PROPN   NNP     proper noun             noun, proper singular
and      CCONJ   CC      coordinating conjunction                conjunction, coordinating
me       PRON    PRP     pronoun                 pronoun, personal
went     VERB    VBD     verb            verb, past tense
to       ADP     IN      adposition              conjunction, subordinating or preposition
the      DET     DT      determiner              determiner
school   NOUN    NN      noun            noun, singular or mass
by       ADP     IN      adposition              conjunction, subordinating or preposition
bus      NOUN    NN      noun            noun, singular or mass
```

```
[11]
doc = nlp("My friend will fly to New York fast and she is staying there for 3 dyas.")

for token in doc:
    print(token.text, "\t", token.pos_, "\t", token.tag_, "\t", spacy.explain(token.pos_), "\t\t", spacy.explain(token.tag_))
```
Python

```
My       PRON    PRP$    pronoun                 pronoun, possessive
friend   NOUN    NN      noun            noun, singular or mass
will     AUX     MD      auxiliary               verb, modal auxiliary
fly      VERB    VB      verb            verb, base form
to       ADP     IN      adposition              conjunction, subordinating or preposition
New      PROPN   NNP     proper noun             noun, proper singular
York     PROPN   NNP     proper noun             noun, proper singular
fast     ADV     RB      adverb                  adverb
and      CCONJ   CC      coordinating conjunction                conjunction, coordinating
she      PRON    PRP     pronoun                 pronoun, personal
is       AUX     VBZ     auxiliary               verb, 3rd person singular present
```

분류를 뜻하는 Classification은 Supervised learning 지도학습의 일종으로 기존에 존재하는 데이터의 Category 관계를 파악하고, 새롭게 관측된 데이터의 Category를 스스로 판별하는 과정이다.

Lema, stem
Lema : 단어의 base form을 찾는 것
Stem : 단어의 어근의 의미
Stem의 예 : university라는 단어의 lema(university),stem(univer)
Stem은 자체로는 완전한 단어의 형태는 아니지만, 의미는 가지고 있다.
Lema Algorithm : 동사적인 문법적인 접근을 통해서 분석하는 방식, dictionary와 같은 vocabulary가 필요
Stem Algorithm : 언어의 문법적인 내용에 대해서는 상관하지 않는 방식, 단어의 suffix나 prefix를 추려낸 이후 처리하는 방식

4주차: 파이썬으로 Database 생성.관리

```
# 파이썬으로 Database 생성
# SQLite3의 경우 connection을 생성할 때, 해당 database 파일이 존재하지 않으면 생성
# 다른 관계형 (Relational) DBMS (Database Management System)의 경우는 사전에 Database가 생성되어
# 있어야 합니다.

import sqlite3

conn = sqlite3.connect("coffee.db")

print("Database 연결")
```

[1]

```
Database 연결
```

```
# Database라는 곳은 데이터들을 보관하는 곳
# table : 데이터들을 기준에 의해서 논리적으로 묶은 것
#         행 (row, record) 및 열 (column, )로 이루어 짐
# table schema (어떤 데이터로 들려 구성이고, 각 열이 어떤 데이터 형식을 가지는지 정의)
# 테이블 생성하고 데이터를 입력

conn.execute('''
CREATE TABLE users (
    id      INT     PRIMARY KEY NOT NULL,
    name    TEXT    NOT NULL
)
''')

print("테이블이 생성되었습니다.")
```

[2]

```
테이블이 생성되었습니다.
```

```
# Database가 존재하고, Table 생성한 상태이니!
# 이제 이 table에 데이터를 입력

conn.execute('''
INSERT INTO users (id, name)
VALUES (1, "김철수")
''')
conn.execute('''
INSERT INTO users (id, name)
VALUES (2, "이영희")
''')
conn.execute('''
INSERT INTO users (id, name)
```

```
    conn.commit()

    print("데이터 입력완료")
```

[5]

```
데이터 입력완료
```

```
    conn.execute('''
    INSERT INTO users (id, name)
    VALUES (5, "김철수")
    ''')
    conn.execute('''
    INSERT INTO users (id, name)
    VALUES (6, "이영희")
    ''')
    conn.execute('''
    INSERT INTO users (id, name)
    VALUES (7, "이동국")
    ''')
    conn.execute('''
    INSERT INTO users (id, name)
    VALUES (8, "안정환")
    ''')

    conn.commit()

    print("데이터 입력완료")
```

[ ]

```
    # 데이터를 보여준다!
    # 우선 데이터를 조회해서 Result Set를 가져온다

    cursor = conn.execute('''
    SELECT *  -- 테이블의 모든 열을 보여달라
    FROM users
    ''')
```

[12]

```
    type(cursor)
```

[7]

```
sqlite3.Cursor
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

tensorflow_nlp.ipynb

NLP : 자연어 처리

1) Classification (분류) - 라벨링이 필요

2) Machine Translation (번역)

3) Text Generation (문장 생성)

4) Voice Assistants (보이스 비서)

==> 이런 자연어처리들을 하기 위해서 필요한 것이 -> sequence prolems!!! (문장을 이루는 단어를 숫자로 표현하는 문제) ==> NLP (Natural Language Processing) : 자연어에서 정보를 추려해는 것인 목적 ==> NLU (Natural Language Understanding) : 자연어를 이해하고 행동이 목적

자연어에서 이야기하는 용어

1. Text
2. Speech

우리가 하려는 과정은

텍스트 -> 이 텍스트를 숫자로 변환 -> 모델 생성 -> 패턴을 찾기 위해 모델을 학습 -> 패턴 이용 (예측)

```
[1]
import tensorflow as tf
print(tf.__version__)
```
Python

```
2.6.0
```

```
[2]
from helper_functions import unzip_data, create_tensorboard_callback, plot_loss_curves, compare_historys
```
Python

```
[3]
unzip_data("nlp_getting_started.zip")
```
Python

```
import pandas as pd
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")

train_df.head(), train_df.shape, test_df.head(), test_df.shape
```

[3] Python

```
(   id keyword location                                          text  \
 0   1     NaN      NaN  Our Deeds are the Reason of this #earthquake M...
 1   4     NaN      NaN             Forest fire near La Ronge Sask. Canada
 2   5     NaN      NaN  All residents asked to 'shelter in place' are ...
 3   6     NaN      NaN  13,000 people receive #wildfires evacuation or...
 4   7     NaN      NaN  Just got sent this photo from Ruby #Alaska as ...

    target
 0       1
 1       1
 2       1
 3       1
 4       1  ,
 (7613, 5),
    id keyword location                                          text
 0   0     NaN      NaN                 Just happened a terrible car crash
 1   2     NaN      NaN  Heard about #earthquake is different cities, s...
 2   3     NaN      NaN  there is a forest fire at spot pond, geese are...
 3   9     NaN      NaN           Apocalypse lighting. #Spokane #wildfires
 4  11     NaN      NaN      Typhoon Soudelor kills 28 in China and Taiwan,
 (3263, 4))
```

```
train_df_shuffled = train_df.sample(frac = 1, random_state=42)
train_df_shuffled.head(), train_df_shuffled.shape
```

[4] Python

```
(        id      keyword               location  \
 2644  3796  destruction                    NaN
 2227  3185       deluge                    NaN
 5448  7769       police                     UK
 132    191   aftershock                    NaN
 6845  9810       trauma  Montgomery County, MD

                                           text  target
 2644  So you have a new weapon that can cause un-ima...       1
 2227  The f$&amp;@ing things I do for #GISHWHES Just...       0
 5448  DT @georgegalloway: RT @Galloway4Mayor: ðÜÏThe...       1
 132   Aftershock back to school kick off was great. ...       0
 6845  in response to trauma Children of Addicts deve...       0  ,
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

우리가 하려는 것

입력 (text 컬럼) -> 머신러닝 알고리즘 -> 출력 (target 컬럼)

```
train_df.target.value_counts()

# 결과값이 0과 1밖에 없기에 binary classification 문제
```

[5]

```
0    4342
1    3271
Name: target, dtype: int64
```

```
print(f"총 학습할 데이터의 수 : {len(train_df)}")
print(f"총 테스트할 데이터의 수 : {len(test_df)}")
print(f"총 데이터의 수 : {len(train_df) + len(test_df)}")
```

[6]

```
총 학습할 데이터의 수 : 7613
총 테스트할 데이터의 수 : 3263
총 데이터의 수 : 10876
```

```
import random

random_index = random.randint(0, len(train_df) - 5)
for row in train_df_shuffled[ ["text", "target"] ][random_index:random_index + 5].itertuples():
    _, text, target = row
    print(f"Target: {target}", "부정" if target > 0 else "긍정")
    print(f"Text: \n{text}\n")
    print("---\n")
```

[7]

```
Target: 1 부정
Text:
Rly tragedy in MP: Some live to recount horror: ÛÏWhen I saw coaches of my train plunging into water I called ... http://t.co/Calk5nv5Vc

---

Target: 1 부정
Text:
? 19th Day Since 17-Jul-2015 -- Nigeria: Suicide Bomb Attacks Killed 64 People; Blamed: Boko Haram [L.A. Times/AP] | http://t.co/O2cdKpSDfp

---

Target: 0 긍정
Text:
well it feels like im on fire.

---

Target: 1 부정
Text:
Suncorp net profit rises to $1.13 billion in worst year of natural disaster claims http://t.co/cwZ37lNDVk

---

Target: 0 긍정
show more (open the raw output data in a text editor) ...

Text:
Want to work at Swedish Health Services? We're #hiring in #Seattle WA! Click for details: http://t.co/4KDThCtEmV #Nursing #Job #Jobs
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
[8]
type(train_df_shuffled["text"]), train_df_shuffled["text"]
```

Python

```
(pandas.core.series.Series,
 2644    So you have a new weapon that can cause un-ima...
 2227    The f$&amp;amp;@ing things I do for #GISHWHES Just...
 5448    DT @georgegalloway: RT @Galloway4Mayor: ÞÛÏThe...
 132     Aftershock back to school kick off was great. ...
 6845    in response to trauma Children of Addicts deve...
                              ...
 5226    @Eganator2000 There aren't many Obliteration s...
 5390    just had a panic attack bc I don't have enough...
 860     Omron HEM-712C Automatic Blood Pressure Monito...
 7603    Officials say a quarantine is in place at an A...
 7270    I moved to England five years ago today. What ...
 Name: text, Length: 7613, dtype: object)
```

```
[9]
type(train_df_shuffled["text"].to_numpy()), train_df_shuffled["text"].to_numpy()
```

Python

```
(numpy.ndarray,
 array(['So you have a new weapon that can cause un-imaginable destruction.',
        'The f$&amp;amp;@ing things I do for #GISHWHES Just got soaked in a deluge going for pads and tampons. Thx @mishacollins @/@',
        'DT @georgegalloway: RT @Galloway4Mayor: \x89ÛÏThe CoL police can catch a pickpocket in Liverpool Stree... http://t.co/vXIn1gOq4Q',
        ...,
        'Omron HEM-712C Automatic Blood Pressure Monitor STANDARD AND LARGE BP CUFFS http://t.co/gJBAInQWN9 http://t.co/jPhgpL1c5x',
        'Officials say a quarantine is in place at an Alabama home over a possible Ebola case after developing symptoms... http://t.co/rqKK15uhEY',
        'I moved to England five years ago today. What a whirlwind of time it has been! http://t.co/eaSlGeA1B7'],
       dtype=object))
```

```
[10]
from sklearn.model_selection import train_test_split

train_sentences, val_sentences, train_labels, val_labels = train_test_split(
    train_df_shuffled["text"].to_numpy(),
    train_df_shuffled["target"].to_numpy(),
    test_size = 0.1,
    random_state= 42
)
```

Python

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
[11]  len(train_sentences), len(train_labels), len(val_sentences), len(val_labels)
                                                                                    Python
...   (6851, 6851, 762, 762)
```

```
[12]  train_sentences[:10], train_labels[:10]
                                                                                    Python
...   (array(['@mogacola @zamtriossu i screamed after hitting tweet',
               'Imagine getting flattened by Kurt Zouma',
               '@Gurmeetramrahim #MSGDoing111WelfareWorks Green S welfare force ke appx 65000 members har time disaster victim ki help ke liye tyar hai....',
               "@shakjn @C7 @Magnums im shaking in fear he's gonna hack the planet",
               'Somehow find you and I collide http://t.co/Ee8RpOahPk',
               '@EvaHanderek @MarleyKnysh great times until the bus driver held us hostage in the mall parking lot lmfao',
               'destroy the free fandom honestly',
               'Weapons stolen from National Guard Armory in New Albany still missing #Gunsense http://t.co/lKNU8902JE',
               '@wfaaweather Pete when will the heat wave pass? Is it really going to be mid month? Frisco Boy Scouts have a canoe trip in Okla.',
               'Patient-reported outcomes in long-term survivors of metastatic colorectal cancer - British Journal of Surgery http://t.co/5Yl4DC1Tqt'],
              dtype=object),
       array([0, 0, 1, 0, 0, 1, 1, 0, 1, 1], dtype=int64))
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

이제 모든 데이터 준비가 끝났습니다!

라벨인 target은 숫자! -> 기계가 아주 잘 인식!!!

문제는 text가 문자열! -> 기계가 인식하기 어려움!!!

문자열을 숫자로 바꾸어 주는 과정이 필요합니다

NLP에서는 텍스트를 숫자로 바꾸어주는 2가지 주요 개념이 존재!

1) Tokenization : 단어, 문자 등을 바로 숫자로 변환

```
1. Word-level tokenization : 모든 기본 단위가 word
2. Chacracter-level tokenization : A ~ Z => 1 ~ 26, 모든 글자가 단위
3. Sub-word-level tokenization : word-level tokenization과 character-level tokenization의 중간 / favorite => favor + rite
```

2) Embedding : 자연어를 학습할 수 있는 형태로 표현한 것, 이런 표현은 feature vector의 형태로 제공합니다.

```
dance라는 단어가 있을 때 [-0.3412, 0.3424, -0.2343, 0.9934, 0.1112]와 같이 표현됨
feature vector의 크기는 조정이 가능하다.
1. 자신만의 embedding을 사용
2. 사전에 학습된 embedding을 사용
```

각각 어떤 것을 사용해야하는 것은 문제에 따라 다르다!

[13]
```python
import tensorflow as tf
# from tensorflow.keras.layers.experimental.preprocessing import TextVectorization # Tensorflow 2.6 이전 방식
```
Python

[14]
```python
text_vectorizer = tf.keras.layers.TextVectorization(
    max_tokens = None,
    standardize = "lower_and_strip_punctuation",
    split = "whitespace",
    ngrams = None,
    output_mode = "int",
    output_sequence_length = None
)
```
Python

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
round(sum(
    [len(i.split()) for i in train_sentences]) / len(train_sentences)
)
```

[15] Python

```
15
```

```
max_vocab_length = 10000
max_length = 15

text_vectorizer = tf.keras.layers.TextVectorization(
    max_tokens = max_vocab_length,
    standardize = "lower_and_strip_punctuation",
    split = "whitespace",
    ngrams = None,
    output_mode = "int",
    output_sequence_length = max_length
)
```

[16] Python

```
sample_sentence = "There's a flood in my street!"
text_vectorizer([sample_sentence])
```

[17] Python

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_16504/2448844681.py in <module>
      1 sample_sentence = "There's a flood in my street!"
----> 2 text_vectorizer([sample_sentence])

c:\20210906\venv\lib\site-packages\keras\engine\base_layer.py in __call__(self, *args, **kwargs)
   1028       with tf.name_scope(name_scope):
   1029         if not self.built:
-> 1030           self._maybe_build(inputs)
   1031
   1032         if self._autocast:

c:\20210906\venv\lib\site-packages\keras\engine\base_layer.py in _maybe_build(self, inputs)
   2657         # operations.
   2658         with tf_utils.maybe_init_scope(self):
-> 2659           self.build(input_shapes)  # pylint:disable=not-callable
   2660       # We must set also ensure that the layer is marked as built, and the build
   2661       # shape is stored since user defined build functions may not be calling

c:\20210906\venv\lib\site-packages\keras\layers\preprocessing\text_vectorization.py in build(self, input_shape)
    413     # the expression needs to evaluate to True in that case.
    414     if self._split is not None:
--> 415       if input_shape.ndims > 1 and not input_shape[-1] == 1:  # pylint: disable=g-comparison-negation
    416         raise RuntimeError(
    417             "When using TextVectorization to tokenize strings, the innermost "
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
[18]
text_vectorizer.adapt(train_sentences)
```
Python

```
[19]
sample_sentence = "There's a flood in my street!"
text_vectorizer([sample_sentence])
```
Python

```
...
<tf.Tensor: shape=(1, 15), dtype=int64, numpy=
array([[264,   3, 232,   4,  13, 698,   0,   0,   0,   0,   0,   0,   0,
          0,   0]], dtype=int64)>
```

```
[20]
sample_sentence = "I love tensorflow"
text_vectorizer([sample_sentence])
```
Python

```
...
<tf.Tensor: shape=(1, 15), dtype=int64, numpy=
array([[  8, 110,   1,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0]], dtype=int64)>
```

```
[21]
sample_sentence = "I Can do TensorFlow"
text_vectorizer([sample_sentence])
```
Python

```
...
<tf.Tensor: shape=(1, 15), dtype=int64, numpy=
array([[ 8, 71, 68,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0]],
      dtype=int64)>
```

```
[22]
random_sentence = random.choice(train_sentences)
print(f"원래 Text : \n{random_sentence}\n\nVector : ")
text_vectorizer([random_sentence])
```
Python

```
...
원래 Text :
RT AbbsWinston: #Zionist #Terrorist Demolish Tire Repair Shop Structure in #Bethlehem
http://t.co/ph2xLI8nVe http://t.co/22fuxHn7El

Vector :
<tf.Tensor: shape=(1, 15), dtype=int64, numpy=
array([[  96, 3199, 2211,  309,  516, 7543, 1610, 2297, 2277,    4, 6109,
          1,    1,    0,    0]], dtype=int64)>
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
[23]  words_in_vocab = text_vectorizer.get_vocabulary()
```
Python

```
[24]  type(words_in_vocab), len(words_in_vocab)
```
Python

```
...  (list, 10000)
```

```
[25]  top_5_words = words_in_vocab[:5]  # [UNK] : unknown 를 의미
      bottom_5_words = words_in_vocab[-5:]
```
Python

```
[26]  print(f"vocab에 있는 단어 총 갯수 : {len(words_in_vocab)}")
      print(f"자주 사용하는 단어 5개 : {top_5_words}")
      print(f"자주 사용하지 않는 단어 5개 : {bottom_5_words}")
```
Python

```
...  vocab에 있는 단어 총 갯수 : 10000
     자주 사용하는 단어 5개 : ['', '[UNK]', 'the', 'a', 'in']
     자주 사용하지 않는 단어 5개 : ['pages', 'paeds', 'pads', 'padres', 'paddytomlinson1']
```

정말 간단하게 tokenization을 했습니다. 문장을 숫자로 이루어진 벡터로 만들었습니다.

embedding의 특징 : tokennization의 경우 tensorflow = 1, I = 8

이 값들은 우리가 다시 새로운 데이터셋으로 adopt를 하지 않으면

계속 유지

embedding은 단어들간의 관계 (relation)를 표현하기 때문에, 학습을 하면서 변경이 된다! 즉 개선이 된다.

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

[27]
```python
from tensorflow.keras import layers
```
Python

[28]
```python
embedding = layers.Embedding(
    input_dim = max_vocab_length,
    output_dim = 128, # embedding vector의 크기
    embeddings_initializer = "uniform",
    input_length = max_length
)
embedding
```
Python

```
<keras.layers.embeddings.Embedding at 0x20937a6d790>
```

[29]
```python
random_sentence = random.choice(train_sentences)
print(f"원래 Text : \n{random_sentence}\n\nEmbedded : ")
sample_embed = embedding(text_vectorizer([random_sentence]))
sample_embed
```
Python

```
원래 Text :
@JWalkerLyle Discovered by @NickCannon
 Listen/Buy @realmandyrain #RIOT on @iTunesMusic @iTunes https://t.co/dehMym5lpk 🇺_ #BlowMandyUp

Embedded :
<tf.Tensor: shape=(1, 15, 128), dtype=float32, numpy=
array([[[-0.01210344, -0.01320591,  0.01990546, ...,  0.02371682,
          0.02373096,  0.02755833],
        [ 0.04074123,  0.04780031, -0.04101676, ...,  0.00250701,
         -0.04857511,  0.01411618],
        [ 0.0349392 , -0.0039593 , -0.00277793, ...,  0.03831022,
          0.0356931 , -0.02017055],
        ...,
        [-0.00223591,  0.03398992,  0.04187554, ...,  0.03936077,
          0.00356712, -0.02351511],
        [ 0.04493793, -0.04360494,  0.00966182, ..., -0.00689634,
         -0.03204142, -0.02886941],
        [ 0.04493793, -0.04360494,  0.00966182, ..., -0.00689634,
         -0.03204142, -0.02886941]]], dtype=float32)>
```

[30]
```python
sample_embed[0][0]
```
Python

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

이제 우리는 학습할 수 있는 데이터를 준비했습니다.

1. Model 0 : Naive Bayes

2. Model 1 : Feed-forward neural network (dense model)

3. Model 2 : LSTM

4. Model 3 : GRU

5. Model 4 : Bidirectional-LSTM

6. Model 5 : 1D CNN

7. Model 6 : Tensorflow Hub pretrained feature extractor

8. Model 7 : 전이학습

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
[31]
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
```
Python

```
[32]
model_0 = Pipeline([
    ("tfidf", TfidfVectorizer()), # tfidf를 사용해서 단어를 숫자로 변경
    ("clf", MultinomialNB()) # 텍스트를 모델링
])
```
Python

```
[33]
model_0.fit(train_sentences, train_labels)
```
Python

```
...
Pipeline(steps=[('tfidf', TfidfVectorizer()), ('clf', MultinomialNB())])
```

```
[34]
baseline_score = model_0.score(val_sentences, val_labels)
print(f"Model 0의 accuracy는 {baseline_score * 100:.2f}%")
```
Python

```
...
Model 0의 accuracy는 79.27%
```

```
[35]
baseline_preds = model_0.predict(val_sentences)
baseline_preds[:20]
```
Python

```
...
array([1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1],
      dtype=int64)
```

```
[36]
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
```
Python

```
def calculate_results(y_true, y_pred):
    model_accuracy = accuracy_score(y_true, y_pred) * 100
    model_precision, model_recall, model_f1, _ = precision_recall_fscore_support(y_true, y_pred, average="weighted")
    model_results = {
        "accuracy" : model_accuracy,
        "precision" : model_precision,
        "recall" : model_recall,
        "f1" : model_f1
    }
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
    return model_results
```
[37] Python

```
baseline_results = calculate_results(y_true = val_labels, y_pred = baseline_preds)
baseline_results
```
[38] Python

```
... {'accuracy': 79.26509186351706,
 'precision': 0.8111390004213173,
 'recall': 0.7926509186351706,
 'f1': 0.7862189758049549}
```

Model 1 : simple Dense Model

```
from helper_functions import create_tensorboard_callback
```
[39] Python

```
SAVE_DIR = "model_logs"
```
[40] Python

```
# Functional API으로 모델을 생성! 이전에는 sequencial 방법을 사용
from tensorflow.keras import layers

inputs = layers.Input(shape=(1,), dtype="string") # 입력이 1차원 문자열
x = text_vectorizer(inputs)
x = embedding(x)
x = layers.GlobalAveragePooling1D()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)

model_1 = tf.keras.Model(inputs, outputs, name = "model_1_dense")
```
[41] Python

```
model_1.compile(
    loss = "binary_crossentropy",
    optimizer = tf.keras.optimizers.Adam(),
    metrics = ["accuracy"]
)
```
[42] Python

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

tensorflow_nlp4_2.ipynb | tensorflow_nlp4.ipynb | tensorflow_nlp3.ipynb | tensorflow_nlp2.ipynb | tensorflow_nlp (1).ipynb | tensorflow_nlp5ipynb

C: > Users > 98040 > Downloads > tensorflow_nlp4_2.ipynb > tensorflow_nlp.ipynb

+ 코드 + Markdown | 모두 실행 | 모든 셀의 출력 지우기 | Outline ... Python 3.8.10 64-bit

[95]
```
model_1.summary()
```
Python

```
Model: "model_1_dense"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 1)]               0
_________________________________________________________________
text_vectorization_1 (TextVe (None, 15)                0
_________________________________________________________________
embedding (Embedding)        (None, 15, 128)           1280000
_________________________________________________________________
global_average_pooling1d_1 ( (None, 128)               0
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,280,129
Trainable params: 1,280,129
Non-trainable params: 0
_________________________________________________________________
```

[96]
```
model_1_history = model_1.fit(
    train_sentences,
    train_labels,
    epochs = 5,
    validation_data = (val_sentences, val_labels),
    callbacks = [
        create_tensorboard_callback(
            dir_name = SAVE_DIR,
            experiment_name = "simple_dense_model"
        )
    ]
)
```
Python

```
Saving TensorBoard log files to: model_logs/simple_dense_model/20210912-145913
Epoch 1/5
215/215 [==============================] - 4s 15ms/step - loss: 0.5257 - accuracy: 0.8181 - val_loss: 0.4973 - val_accuracy: 0.7835
Epoch 2/5
215/215 [==============================] - 2s 10ms/step - loss: 0.3420 - accuracy: 0.8860 - val_loss: 0.4614 - val_accuracy: 0.7900
Epoch 3/5
215/215 [==============================] - 2s 10ms/step - loss: 0.2707 - accuracy: 0.9021 - val_loss: 0.4607 - val_accuracy: 0.7927
```

Jupyter Server: local

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

tensorflow_nlp4_2.ipynb | tensorflow_nlp4.ipynb | tensorflow_nlp3.ipynb | tensorflow_nlp2.ipynb | tensorflow_nlp (1).ipynb | tensorflow_nlp5ipynb

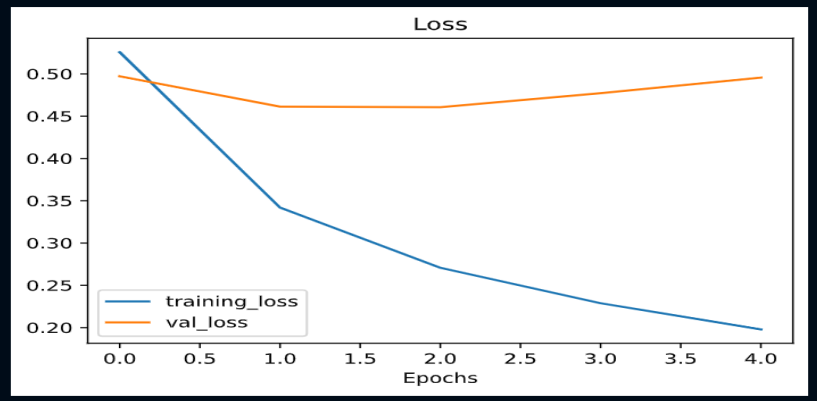
C: > Users > 98040 > Downloads > tensorflow_nlp4_2.ipynb > tensorflow_nlp.ipynb

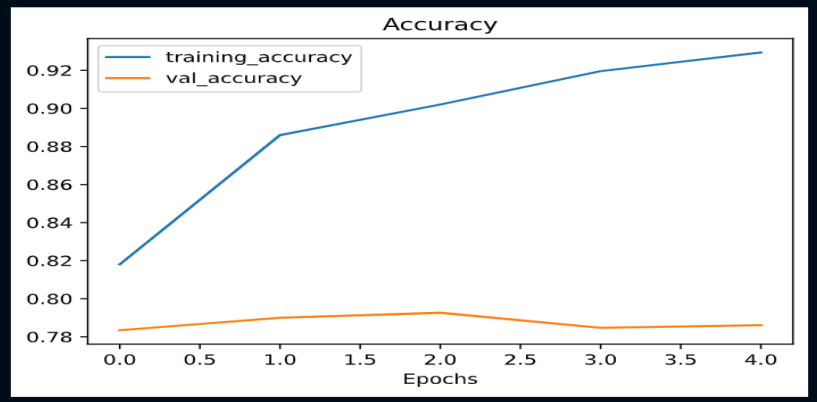
+ 코드 + Markdown | 모두 실행 | 모든 셀의 출력 지우기 | Outline ... Python 3.8.10 64-bit

```
215/215 [==============================] - 2s 10ms/step - loss: 0.1979 - accuracy: 0.9294 - val_loss: 0.4955 - val_accuracy: 0.7861
```

```
from helper_functions import plot_loss_curves

plot_loss_curves(model_1_history)
```

[97] Python





Jupyter Server: local

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

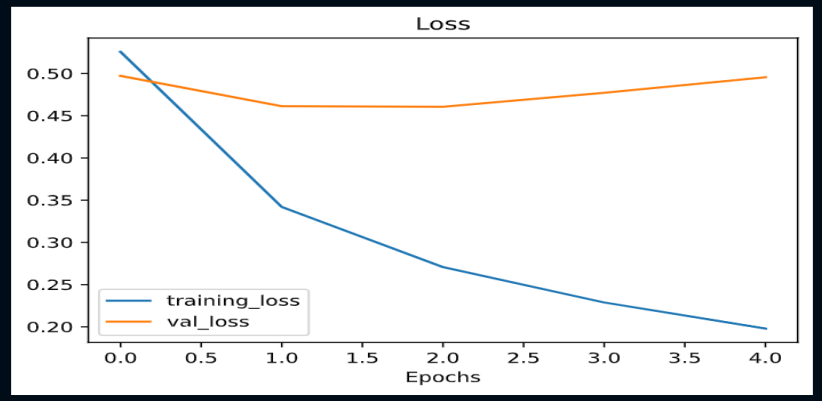
tensorflow_nlp4_2.ipynb | tensorflow_nlp4.ipynb | tensorflow_nlp3.ipynb | tensorflow_nlp2.ipynb | tensorflow_nlp (1).ipynb | tensorflow_nlp5ipynb

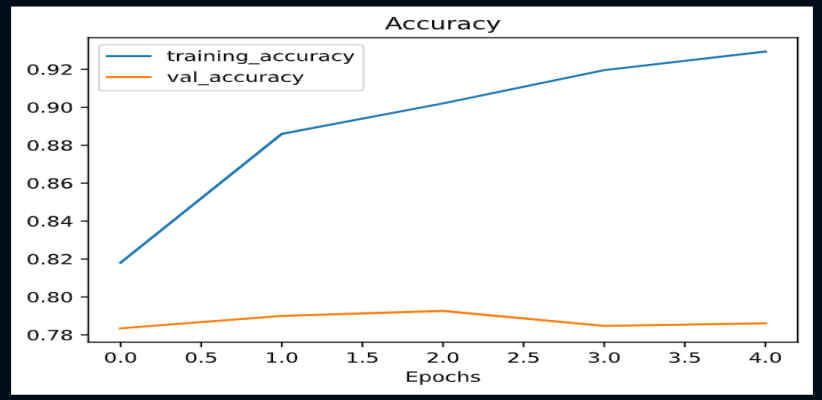
C: > Users > 98040 > Downloads > tensorflow_nlp4_2.ipynb > tensorflow_nlp.ipynb

+ 코드 + Markdown | 모두 실행 | 모든 셀의 출력 지우기 | Outline ···

Python 3.8.10 64-bit

[98]
```
plot_loss_curves(model_1_history)  # pooling layer가 없을 때
```
Python






```
model_1_pred_probs = model_1.predict(val_sentences)
```

Jupyter Server: local

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

tensorflow_nlp4_2.ipynb | tensorflow_nlp4.ipynb | tensorflow_nlp3.ipynb | tensorflow_nlp2.ipynb | tensorflow_nlp (1).ipynb | tensorflow_nlp5ipynb

C: > Users > 98040 > Downloads > tensorflow_nlp4_2.ipynb > tensorflow_nlp.ipynb

+ 코드 + Markdown | 모두 실행 | 모든 셀의 출력 지우기 | Outline ... | Python 3.8.10 64-bit

[99]
```
model_1_pred_probs = model_1.predict(val_sentences)
model_1_pred_probs[:10], type(model_1_pred_probs), model_1_pred_probs.shape
```
Python

```
(array([[0.36688352],
        [0.7753926 ],
        [0.99809843],
        [0.10293064],
        [0.10197628],
        [0.9463314 ],
        [0.90672576],
        [0.99573153],
        [0.97389305],
        [0.27424967]], dtype=float32),
 numpy.ndarray,
 (762, 1))
```

[100]
```
model_1_preds = tf.squeeze(tf.round(model_1_pred_probs))
model_1_preds[:20]
```
Python

```
<tf.Tensor: shape=(20,), dtype=float32, numpy=
array([0., 1., 1., 0., 0., 1., 1., 1., 1., 0., 0., 1., 0., 0., 0., 0., 0.,
       0., 0., 1.], dtype=float32)>
```

[101]
```
model_1_results = calculate_results(y_true = val_labels, y_pred=model_1_preds)
model_1_results
```
Python

```
{'accuracy': 78.60892388451444,
 'precision': 0.7907394181632303,
 'recall': 0.7860892388451444,
 'f1': 0.7831536805930754}
```

[102]
```
import numpy as np

np.array( list(model_1_results.values())) > np.array( list(baseline_results.values()))
```
Python

```
array([False, False, False, False])
```

```
words_in_vocab = text_vectorizer.get_vocabulary()
```

Jupyter Server: local

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

tensorflow_nlp4_2.ipynb | tensorflow_nlp4.ipynb | tensorflow_nlp3.ipynb | tensorflow_nlp2.ipynb | tensorflow_nlp (1).ipynb | tensorflow_nlp5ipynb

C: > Users > 98040 > Downloads > tensorflow_nlp4_2.ipynb > tensorflow_nlp.ipynb

+ 코드 + Markdown | 모두 실행 | 모든 셀의 출력 지우기 | Outline ··· Python 3.8.10 64-bit

[103]
```
words_in_vocab = text_vectorizer.get_vocabulary()
len(words_in_vocab), words_in_vocab[:10]
```
Python

```
(10000, ['', '[UNK]', 'the', 'a', 'in', 'to', 'of', 'and', 'i', 'is'])
```

[104]
```
model_1.summary()
```
Python

```
Model: "model_1_dense"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 1)]               0
_________________________________________________________________
text_vectorization_1 (TextVe (None, 15)                0
_________________________________________________________________
embedding (Embedding)        (None, 15, 128)           1280000
_________________________________________________________________
global_average_pooling1d_1 ( (None, 128)               0
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,280,129
Trainable params: 1,280,129
Non-trainable params: 0
_________________________________________________________________
```

[105]
```
embed_weights = model_1.get_layer("embedding").get_weights()[0]
embed_weights, type(embed_weights), len(embed_weights)
```
Python

```
(array([[ 0.01388992,  0.00923975, -0.00253602, ..., -0.03635037,
         -0.09055129, -0.10382263],
        [-0.014756  , -0.03807469, -0.03088848, ..., -0.03337269,
         -0.06784556, -0.04919549],
        [ 0.02840471, -0.05791222,  0.04579973, ...,  0.04808735,
         -0.07763419, -0.07451463],
        ...,
        [-0.04527675,  0.04225541,  0.01560595, ...,  0.01072598,
         -0.01862408,  0.0406022 ],
```

Jupyter Server: local

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

tensorflow_nlp4_2.ipynb | tensorflow_nlp4.ipynb | tensorflow_nlp3.ipynb | tensorflow_nlp2.ipynb | tensorflow_nlp (1).ipynb | tensorflow_nlp5ipynb

C: > Users > 98040 > Downloads > tensorflow_nlp4_2.ipynb > tensorflow_nlp.ipynb

+ 코드 + Markdown | 모두 실행 | 모든 셀의 출력 지우기 | Outline ... Python 3.8.10 64-bit

```
        -0.15528251, -0.15540262]], dtype=float32),
 numpy.ndarray,
 10000)
```

[106]
```
embed_weights, type(embed_weights), embed_weights.shape
```
Python

```
... (array([[ 0.01388992,  0.00923975, -0.00253602, ..., -0.03635037,
         -0.09055129, -0.10382263],
        [-0.014756  , -0.03807469, -0.03088848, ..., -0.03337269,
         -0.06784556, -0.04919549],
        [ 0.02840471, -0.05791222,  0.04579973, ...,  0.04808735,
         -0.07763419, -0.07451463],
        ...,
        [-0.04527675,  0.04225541,  0.01560595, ...,  0.01072598,
         -0.01862408,  0.0406022 ],
        [-0.06461954, -0.07283572, -0.02630498, ..., -0.02837508,
         -0.09114114, -0.11586291],
        [-0.10019533, -0.12281165, -0.06588618, ..., -0.14001685,
         -0.15528251, -0.15540262]], dtype=float32),
 numpy.ndarray,
 (10000, 128))
```

[107]
```
import io

out_v = io.open("embeddig_vectors.tsv", "w", encoding = "utf-8")
out_m = io.open("embeddig_metadata.tsv", "w", encoding = "utf-8")

for num, word in enumerate(words_in_vocab):
    if num == 0:
        continue
    vec = embed_weights[num]
    out_m.write(word + "\n")
    out_v.write("\t".join([str(x) for x in vec]) + "\n")
out_v.close()
out_m.close()
```
Python

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

Recurrent Neural Network (RNN)

기본 개념 : 미래를 예측하기 위해 과거의 정보를 활용한다.

입력(X) -> 이전 입력을 기반으로 계산 -> 출력(y)

트위터와 같이 짧은 구절들을 다룰 때 유용

독해할 때 모르는 단어가 나올 때 이전 문맥을 통해 그 단어를 유추하라.

Massive earthquake last week, no? (엄청한 지진이 있었다)

No massive earthquake last week. (지진이 없었다)

1) one to one : 입력 1개, 출력도 1개

2) one to many : 입력 1개, 출력은 many

3) Many to one : 입력 many, 출력은 1개 (텍스트 분류)

4) Many to many : 입력 many, 출력은 many (머신 번역 이나 STT)

RNN에서 발전하는 개념들 : Long short-term memory cells (LSTMs), Gated recurrent units (GRUs), Bidirectional RNN

```python
# Model 2 : LSTM

from tensorflow.keras import layers

inputs = layers.Input(shape=(1, ), dtype="string")
x = text_vectorizer(inputs)
x = embedding(x)
print(x.shape)
x = layers.LSTM(64)(x)
print(x.shape)
outputs = layers.Dense(1, activation="sigmoid")(x)
model_2 = tf.keras.Model(inputs, outputs, name = "model_2_LSTM")
```

[54]

Python

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
(None, 15, 128)
(None, 64)
```

```
model_2.compile(
    loss = "binary_crossentropy",
    optimizer = tf.keras.optimizers.Adam(),
    metrics = ["accuracy"]
)
```

[55] Python

```
model_2.summary()
```

[57] Python

```
Model: "model_2_LSTM"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 1)]               0
_________________________________________________________________
text_vectorization_1 (TextVe (None, 15)                0
_________________________________________________________________
embedding (Embedding)        (None, 15, 128)           1280000
_________________________________________________________________
lstm (LSTM)                  (None, 64)                49408
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 1,329,473
Trainable params: 1,329,473
Non-trainable params: 0
_________________________________________________________________
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
model_2_history = model_2.fit(
    train_sentences,
    train_labels,
    epochs = 5,
    validation_data = (val_sentences, val_labels),
    callbacks = [
        create_tensorboard_callback(
            dir_name = SAVE_DIR,
            experiment_name = "LSTM"
        )
    ]
)
```

11] Python

```
Saving TensorBoard log files to: model_logs/LSTM/20210912-145937
Epoch 1/5
215/215 [==============================] - 5s 17ms/step - loss: 0.1868 - accuracy: 0.9369 - val_loss: 0.5740 - val_accuracy: 0.7808
Epoch 2/5
215/215 [==============================] - 2s 11ms/step - loss: 0.1271 - accuracy: 0.9531 - val_loss: 0.6680 - val_accuracy: 0.7717
Epoch 3/5
215/215 [==============================] - 2s 11ms/step - loss: 0.1053 - accuracy: 0.9597 - val_loss: 0.8690 - val_accuracy: 0.7756
Epoch 4/5
215/215 [==============================] - 2s 11ms/step - loss: 0.0899 - accuracy: 0.9664 - val_loss: 0.8168 - val_accuracy: 0.7743
Epoch 5/5
215/215 [==============================] - 2s 11ms/step - loss: 0.0785 - accuracy: 0.9680 - val_loss: 1.1331 - val_accuracy: 0.7822
```

```
model_2_pred_probs = model_2.predict(val_sentences)
model_2_pred_probs.shape, model_2_pred_probs[:10]
```

12] Python

```
((762, 1),
 array([[5.9865415e-03],
        [5.7472461e-01],
        [9.9988490e-01],
        [8.8041127e-03],
        [3.4803152e-04],
        [9.9983895e-01],
        [9.8724520e-01],
        [9.9993277e-01],
        [9.9990714e-01],
        [4.0324411e-01]], dtype=float32))
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

tensorflow_nlp4_2.ipynb | tensorflow_nlp4.ipynb | tensorflow_nlp3.ipynb | tensorflow_nlp2.ipynb | tensorflow_nlp (1).ipynb | tensorflow_nlp5ipynb

C: > Users > 98040 > Downloads > tensorflow_nlp4_2.ipynb > tensorflow_nlp.ipynb

+ 코드 + Markdown | 모두 실행 | 모든 셀의 출력 지우기 | Outline ... Python 3.8.10 64-bit

[113]
```
model_2_pred = tf.squeeze(tf.round(model_2_pred_probs))
model_2_pred[:10]
```
Python

```
<tf.Tensor: shape=(10,), dtype=float32, numpy=array([0., 1., 1., 0., 0., 1., 1., 1., 1., 0.], dtype=float32)>
```

[114]
```
model_2_results = calculate_results(y_true = val_labels, y_pred = model_2_pred)
model_2_results
```
Python

```
{'accuracy': 78.21522309711287,
 'precision': 0.7829133858267716,
 'recall': 0.7821522309711286,
 'f1': 0.7807494233675335}
```

[115]
```
np.array( list(model_2_results.values())) > np.array( list(baseline_results.values()))
```
Python

```
array([False, False, False, False])
```

[116]
```
# GRU

from tensorflow.keras import layers

inputs = layers.Input(shape=(1, ), dtype="string")
x = text_vectorizer(inputs)
x = embedding(x)
print(x.shape)
x = layers.GRU(64)(x)
print(x.shape)
outputs = layers.Dense(1, activation="sigmoid")(x)
model_3 = tf.keras.Model(inputs, outputs, name = "model_3_GRU")
```
Python

```
(None, 15, 128)
(None, 64)
```

[117]
```
model_3.compile(
    loss = "binary_crossentropy",
    optimizer = tf.keras.optimizers.Adam(),
    metrics = ["accuracy"]
)
```
Python

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

[118]
```
model_3.summary()
```
Python

```
Model: "model_3_GRU"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 1)]               0
_________________________________________________________________
text_vectorization_1 (TextVe (None, 15)                0
_________________________________________________________________
embedding (Embedding)        (None, 15, 128)           1280000
_________________________________________________________________
gru (GRU)                    (None, 64)                37248
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 65
=================================================================
Total params: 1,317,313
Trainable params: 1,317,313
Non-trainable params: 0
_________________________________________________________________
```

[119]
```
model_3_history = model_3.fit(
    train_sentences,
    train_labels,
    epochs = 5,
    validation_data = (val_sentences, val_labels),
    callbacks = [
        create_tensorboard_callback(
            dir_name = SAVE_DIR,
            experiment_name = "LSTM"
        )
    ]
)
```
Python

```
Saving TensorBoard log files to: model_logs/LSTM/20210912-145952
Epoch 1/5
215/215 [==============================] - 5s 17ms/step - loss: 0.1479 - accuracy: 0.9448 - val_loss: 0.7973 - val_accuracy: 0.7795
Epoch 2/5
215/215 [==============================] - 2s 11ms/step - loss: 0.0832 - accuracy: 0.9683 - val_loss: 0.8190 - val_accuracy: 0.7756
Epoch 3/5
215/215 [==============================] - 2s 11ms/step - loss: 0.0675 - accuracy: 0.9737 - val_loss: 0.9084 - val_accuracy: 0.7730
Epoch 4/5
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
[120]
model_3_pred_probs = model_3.predict(val_sentences)
model_3_pred_probs.shape, model_3_pred_probs[:10]
```
Python

```
... ((762, 1),
 array([[3.3653080e-03],
        [6.0248554e-01],
        [9.9969208e-01],
        [2.4313271e-02],
        [1.0227686e-04],
        [9.9973130e-01],
        [6.1218250e-01],
        [9.9989116e-01],
        [9.9986011e-01],
        [7.4662399e-01]], dtype=float32))
```

```
[121]
model_3_pred = tf.squeeze(tf.round(model_3_pred_probs))
model_3_pred[:10]
```
Python

```
... <tf.Tensor: shape=(10,), dtype=float32, numpy=array([0., 1., 1., 0., 0., 1., 1., 1., 1., 1.], dtype=float32)>
```

```
[122]
model_3_results = calculate_results(y_true = val_labels, y_pred = model_3_pred)
model_3_results
```
Python

```
... {'accuracy': 77.03412073490814,
 'precision': 0.7718252603398367,
 'recall': 0.7703412073490814,
 'f1': 0.7683227325217538}
```

```
[123]
np.array( list(model_3_results.values())) > np.array( list(baseline_results.values()))
```
Python

```
... array([False, False, False, False])
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```python
# Model 4 : Bidirectional RNN

from tensorflow.keras import layers

inputs = layers.Input(shape=(1, ), dtype="string")
x = text_vectorizer(inputs)
x = embedding(x)
x = layers.Bidirectional(layers.LSTM(64))(x) #
outputs = layers.Dense(1, activation="sigmoid")(x)
model_4 = tf.keras.Model(inputs, outputs, name = "model_4_Bidirectional")
```

[124] Python

```python
model_4.compile(
    loss = "binary_crossentropy",
    optimizer = tf.keras.optimizers.Adam(),
    metrics = ["accuracy"]
)
```

[125] Python

```python
model_4.summary()
```

[126] Python

```
Model: "model_4_Bidirectional"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 1)]               0
_________________________________________________________________
text_vectorization_1 (TextVe (None, 15)                0
_________________________________________________________________
embedding (Embedding)        (None, 15, 128)           1280000
_________________________________________________________________
bidirectional (Bidirectional (None, 128)               98816
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 129
=================================================================
Total params: 1,378,945
Trainable params: 1,378,945
Non-trainable params: 0
_________________________________________________________________
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
[127]
model_4_history = model_4.fit(
    train_sentences,
    train_labels,
    epochs = 5,
    validation_data = (val_sentences, val_labels),
    callbacks = [
        create_tensorboard_callback(
            dir_name = SAVE_DIR,
            experiment_name = "bidirectional_RNN"
        )
    ]
)
```

Python

```
Saving TensorBoard log files to: model_logs/bidirectional_RNN/20210912-150008
Epoch 1/5
215/215 [==============================] - 8s 25ms/step - loss: 0.1006 - accuracy: 0.9666 - val_loss: 1.0281 - val_accuracy: 0.7769
Epoch 2/5
215/215 [==============================] - 3s 15ms/step - loss: 0.0537 - accuracy: 0.9772 - val_loss: 1.1420 - val_accuracy: 0.7808
Epoch 3/5
215/215 [==============================] - 3s 15ms/step - loss: 0.0487 - accuracy: 0.9771 - val_loss: 1.3304 - val_accuracy: 0.7743
Epoch 4/5
215/215 [==============================] - 4s 16ms/step - loss: 0.0424 - accuracy: 0.9797 - val_loss: 1.2581 - val_accuracy: 0.7743
Epoch 5/5
215/215 [==============================] - 3s 16ms/step - loss: 0.0424 - accuracy: 0.9804 - val_loss: 1.3635 - val_accuracy: 0.7625
```

```
[128]
model_4_pred_probs = model_4.predict(val_sentences)
model_4_pred_probs.shape, model_4_pred_probs[:10]
```

Python

```
((762, 1),
 array([[2.0437241e-03],
        [8.6961031e-01],
        [9.9995756e-01],
        [1.1893985e-01],
        [7.7569195e-05],
        [9.9986196e-01],
        [9.5483685e-01],
        [9.9997085e-01],
        [9.9995917e-01],
        [9.9585956e-01]], dtype=float32))
```

```
model_4_preds = tf.squeeze(tf.round(model_4_pred_probs))
```

Jupyter Server: local

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
[129]
model_4_preds = tf.squeeze(tf.round(model_4_pred_probs))
model_4_preds[:10]
```

Python

```
<tf.Tensor: shape=(10,), dtype=float32, numpy=array([0., 1., 1., 0., 0., 1., 1., 1., 1., 1.], dtype=float32)>
```

```
[130]
model_4_results = calculate_results(val_labels, model_4_preds)
model_4_results
```

Python

```
{'accuracy': 76.24671916010499,
 'precision': 0.7621236842389323,
 'recall': 0.7624671916010499,
 'f1': 0.7617099217132606}
```

```
[131]
np.array( list(model_4_results.values())) > np.array( list(baseline_results.values()))
```

Python

```
array([False, False, False, False])
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

CNN (convolutional neural network) for text

Computer Vision에서와 Natural Language Processing에서의 CNN의 큰 차이점

=> 데이터 shape : 이미지는 보통 2차원 (높이와 너비) / 텍스트 즉, sequence는 1차원 (문자열)

일반적인 CNN 구조로 분석하는 과정

입력 -> 토큰화 -> 임베딩 -> 계층 -> 출력

```
[132]
embedding_test = embedding(text_vectorizer(["이것은 첫 문장입니다"]))
conv_1d = layers.Conv1D(filters = 32, kernel_size = 5, activation = "relu")
conv_1d_output = conv_1d(embedding_test)
max_pool = layers.GlobalMaxPool1D()
max_pool_output = max_pool(conv_1d_output)
embedding_test.shape, conv_1d_output.shape, max_pool_output.shape
```

Python

```
(TensorShape([1, 15, 128]), TensorShape([1, 11, 32]), TensorShape([1, 32]))
```

```
[133]
embedding_test[:1], conv_1d_output[:1], max_pool_output[:1]
```

Python

```
(<tf.Tensor: shape=(1, 15, 128), dtype=float32, numpy=
 array([[[-0.01226095,  0.00564478, -0.03022191, ..., -0.06127714,
          -0.059826  , -0.03393448],
         [-0.01226095,  0.00564478, -0.03022191, ..., -0.06127714,
          -0.059826  , -0.03393448],
         [-0.01226095,  0.00564478, -0.03022191, ..., -0.06127714,
          -0.059826  , -0.03393448],
         ...,
         [-0.00932427, -0.00481054, -0.02803688, ..., -0.02540169,
          -0.05236702, -0.09680413],
         [-0.00932427, -0.00481054, -0.02803688, ..., -0.02540169,
          -0.05236702, -0.09680413],
         [-0.00932427, -0.00481054, -0.02803688, ..., -0.02540169,
          -0.05236702, -0.09680413]]], dtype=float32)>,
 <tf.Tensor: shape=(1, 11, 32), dtype=float32, numpy=
 array([[[0.00177524, 0.        , 0.01903918, 0.        , 0.        ,
          0.02378732, 0.        , 0.02839861, 0.        , 0.01463415,
          0.00041108, 0.07059558, 0.        , 0.        , 0.        ,
          0.        , 0.02316282, 0.06059223, 0.        , 0.        ,
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```python
# Model 5 : CNN for Text

from tensorflow.keras import layers

inputs = layers.Input(shape=(1, ), dtype="string")
x = text_vectorizer(inputs)
x = embedding(x)

x = layers.Conv1D(filters = 32, kernel_size = 5, activation = "relu")(x)
x = layers.GlobalMaxPooling1D()(x)

outputs = layers.Dense(1, activation="sigmoid")(x)
model_5 = tf.keras.Model(inputs, outputs, name = "model_5_Conv1D")
```

[134] Python

```python
model_5.compile(
    loss = "binary_crossentropy",
    optimizer = tf.keras.optimizers.Adam(),
    metrics = ["accuracy"]
)
```

[135] Python

```python
model_5.summary()
```

[136] Python

```
Model: "model_5_Conv1D"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         [(None, 1)]               0
_________________________________________________________________
text_vectorization_1 (TextVe (None, 15)                0
_________________________________________________________________
embedding (Embedding)        (None, 15, 128)           1280000
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 11, 32)            20512
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 32)                0
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 33
=================================================================
Total params: 1,300,545
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
model_5_history = model_5.fit(
    train_sentences,
    train_labels,
    epochs = 5,
    validation_data = (val_sentences, val_labels),
    callbacks = [
        create_tensorboard_callback(
            dir_name = SAVE_DIR,
            experiment_name = "Conv1D"
        )
    ]
)
```

[137] Python

```
Saving TensorBoard log files to: model_logs/Conv1D/20210912-150030
Epoch 1/5
215/215 [==============================] - 4s 15ms/step - loss: 0.1319 - accuracy: 0.9537 - val_loss: 0.8450 - val_accuracy: 0.7782
Epoch 2/5
215/215 [==============================] - 2s 10ms/step - loss: 0.0802 - accuracy: 0.9704 - val_loss: 0.9774 - val_accuracy: 0.7730
Epoch 3/5
215/215 [==============================] - 2s 10ms/step - loss: 0.0637 - accuracy: 0.9762 - val_loss: 1.0877 - val_accuracy: 0.7677
Epoch 4/5
215/215 [==============================] - 2s 10ms/step - loss: 0.0578 - accuracy: 0.9775 - val_loss: 1.1608 - val_accuracy: 0.7730
Epoch 5/5
215/215 [==============================] - 2s 9ms/step - loss: 0.0520 - accuracy: 0.9809 - val_loss: 1.2063 - val_accuracy: 0.7598
```

```
model_5_pred_probs = model_5.predict(val_sentences)
model_5_pred_probs[:10]
```

[138] Python

```
array([[1.1564916e-01],
       [9.4155478e-01],
       [9.9985158e-01],
       [1.9753844e-02],
       [3.1371183e-08],
       [9.9725163e-01],
       [9.8498249e-01],
       [9.9997097e-01],
       [9.9999845e-01],
       [8.3846498e-01]], dtype=float32)
```

```
model_5_preds = tf.squeeze(tf.round(model_5_pred_probs))
model_5_preds[:10]
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
...  <tf.Tensor: shape=(10,), dtype=float32, numpy=array([0., 1., 1., 0., 0., 1., 1., 1., 1., 1.], dtype=float32)>
```

```
[140]  model_5_results = calculate_results(y_true = val_labels, y_pred = model_5_preds)
       model_5_results
```

Python

```
...  {'accuracy': 75.98425196850394,
      'precision': 0.7601337255372377,
      'recall': 0.7598425196850394,
      'f1': 0.7583542786286791}
```

```
[141]  np.array( list(model_5_results.values())) > np.array( list(baseline_results.values()))
```

Python

```
...  array([False, False, False, False])
```

Pretrained (사전에 학습된) 모델을 사용하는 방법 (Transfer Learning for NLP)

Model 0 ~ 5 : 우리가 직접 계층을 만들고 학습을 시켰습니다.

이제 Transfer Learning을 하려고 합니다 : 남들이 잘 만들어놓은 딥러닝 모델을 사용할 수 있다!

어떤 특정한 패턴을 잘 찾는 모델을 사용해서 우리의 패턴을 찾도록 우리의 데이터셋을 적용해보는 것!

```
[48]  !pip install tensorflow_hub
```

Python

```
...  Requirement already satisfied: tensorflow_hub in .\venv\lib\site-packages (0.12.0)
     Requirement already satisfied: protobuf>=3.8.0 in .\venv\lib\site-packages (from tensorflow_hub) (3.17.3)
     Requirement already satisfied: numpy>=1.12.0 in .\venv\lib\site-packages (from tensorflow_hub) (1.19.5)
     Requirement already satisfied: six>=1.9 in .\venv\lib\site-packages (from protobuf>=3.8.0->tensorflow_hub) (1.15.0)
     WARNING: You are using pip version 21.1.3; however, version 21.2.4 is available.
     You should consider upgrading via the 'c:\lecture_tensorflow\venv\scripts\python.exe -m pip install --upgrade pip' command.
```

```
[49]  import tensorflow_hub as hub

      embed = hub.load("https://tfhub.dev/google/universal-sentence-encoder/4")
```

Python

Jupyter Server: local

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
[51]
sample_sentence = "There's a flood in my street!"
text_vectorizer([sample_sentence])
```

Python

```
<tf.Tensor: shape=(1, 15), dtype=int64, numpy=
array([[264,   3, 232,   4,  13, 698,   0,   0,   0,   0,   0,   0,   0,
          0,   0]], dtype=int64)>
```

```
[54]
embed_samples = embed([
    sample_sentence,
    "I love tensorflow"
])
type(embed_samples), embed_samples.shape
```

Python

```
(tensorflow.python.framework.ops.EagerTensor, TensorShape([2, 512]))
```

```
[55]
embed_samples[0]
```

Python

```
<tf.Tensor: shape=(512,), dtype=float32, numpy=
array([-0.01157023,  0.02485909,  0.02878047, -0.01271501,  0.03971536,
        0.08827761,  0.02680985,  0.05589838, -0.01068733, -0.00597294,
        0.00639323, -0.01819517,  0.00030813,  0.09105889,  0.05874642,
       -0.03180626,  0.01512474, -0.05162928,  0.00991366, -0.06865346,
       -0.04209306,  0.0267898 ,  0.0301101 ,  0.00321068, -0.00337969,
       -0.04787357,  0.0226672 , -0.00985927, -0.04063615, -0.01292092,
       -0.04666385,  0.05630299, -0.03949255,  0.00517685,  0.02495829,
       -0.07014439,  0.02871509,  0.0494768 , -0.00633979, -0.08960192,
        0.02807119, -0.00808363, -0.013606  ,  0.0599865 , -0.10361788,
       -0.05195374,  0.00232954, -0.0233253 , -0.03758108,  0.03327729,
       -0.00430604, -0.05894249, -0.06101276, -0.02220005, -0.01575761,
       -0.00474415, -0.03515062, -0.04440377, -0.04174354,  0.04943502,
       -0.0274093 , -0.04133106,  0.01634345, -0.0371368 ,  0.07395209,
       -0.00659237, -0.06943312, -0.00361748,  0.08656701,  0.0771738 ,
       -0.06633057,  0.02086627,  0.06676425,  0.00982106,  0.01410713,
        0.07672231,  0.05026637,  0.02969931, -0.01391874,  0.06082515,
        0.08967175, -0.01739941, -0.02335796,  0.01980951, -0.05318438,
       -0.00597195, -0.01789535,  0.03518632,  0.03316378, -0.01078376,
       -0.0424628 ,  0.02813077, -0.04440409,  0.02138846, -0.07827793,
        0.0037878 ,  0.0211333 ,  0.0489828 ,  0.01862645,  0.04409023,
       -0.00877463, -0.08009899, -0.06491724,  0.03673805,  0.02910935,
        0.04377059, -0.02771871, -0.00020969, -0.04724235,  0.00211793,
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
embed_samples[1]
```

[56] Python

```
<tf.Tensor: shape=(512,), dtype=float32, numpy=
array([ 0.05399965, -0.06587098, -0.03254662, -0.05942422,  0.0079642 ,
       -0.06761236, -0.06450988, -0.00275825,  0.00963327,  0.07514845,
       -0.01024038,  0.05769584,  0.04131394,  0.07075669, -0.02744576,
        0.08019906, -0.00697869, -0.06508592, -0.01986441, -0.05492525,
       -0.0015816 , -0.00790865,  0.04948948,  0.05202748,  0.05732505,
        0.07194925, -0.0433432 , -0.02204052, -0.04422427, -0.02802071,
        0.02626315, -0.01090789, -0.00098572,  0.00480489, -0.05516028,
       -0.04715879, -0.010956  , -0.03169326, -0.01418679,  0.04703688,
        0.0453217 ,  0.0646871 , -0.02817041,  0.02899571,  0.03581711,
        0.07334174, -0.06419893, -0.01024344,  0.06866936,  0.06951907,
       -0.06022416, -0.06350277, -0.01747772,  0.02848996, -0.06284875,
       -0.0663275 ,  0.02705293,  0.072338  ,  0.02619733,  0.00573052,
       -0.07897921, -0.03160059, -0.02693138,  0.04681823, -0.01862348,
        0.0026347 , -0.0615541 ,  0.05306089, -0.06506991,  0.03198494,
       -0.02506077, -0.04198157, -0.04990285,  0.04860826,  0.02608881,
       -0.03703956,  0.03899392,  0.06830154,  0.06992087,  0.06215985,
       -0.05676541, -0.02824926,  0.02718087,  0.02843787,  0.0080534 ,
        0.00514478, -0.02757843, -0.07203251,  0.04293728, -0.03302659,
       -0.04201088,  0.00627255,  0.00981413,  0.07647102, -0.0418903 ,
        0.03871724, -0.02481955,  0.02504495,  0.05058038, -0.0559503 ,
        0.04541542, -0.01264421, -0.00061642,  0.07493638,  0.00837491,
       -0.00631492, -0.00381613,  0.00879551, -0.04895917, -0.01810148,
       -0.00563372, -0.0194271 , -0.06643605, -0.01375432, -0.04244211,
       -0.031002  , -0.01434744, -0.00065672, -0.01838249,  0.04257469,

show more (open the raw output data in a text editor) ...

        0.06106787, -0.02926044,  0.0542987 ,  0.06052126, -0.05219182,
       -0.01690842,  0.07398193, -0.03742874,  0.03452308,  0.05847615,
       -0.00386533, -0.0130894 ,  0.01076577,  0.0173107 ,  0.00526986,
       -0.0067213 ,  0.05337993,  0.0113294 ,  0.05092738, -0.03460051,
        0.05583626,  0.00213643, -0.04915176, -0.0120113 ,  0.07745957,
       -0.03356914, -0.07428339], dtype=float32)>
```

```
sentence_encoder_layer = hub.KerasLayer(
    "https://tfhub.dev/google/universal-sentence-encoder/4",
    input_shape = [],
    dtype = tf.string,
    trainable = False,
    name = "USE"
)
```

Jupyter Server: local

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
model_6 = tf.keras.Sequential([
    sentence_encoder_layer,
    layers.Dense(64, activation = "relu"),
    layers.Dense(1, activation = "sigmoid")
])

model_6.compile(
    loss = "binary_crossentropy",
    optimizer = tf.keras.optimizers.Adam(),
    metrics = ["accuracy"]
)

model_6.summary()
```

[63] Python

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
USE (KerasLayer)             (None, 512)               256797824
_________________________________________________________________
dense_5 (Dense)              (None, 64)                32832
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 65
=================================================================
Total params: 256,830,721
Trainable params: 32,897
Non-trainable params: 256,797,824
_________________________________________________________________
```

```
model_6_history = model_6.fit(
    train_sentences,
    train_labels,
    epochs = 5,
    validation_data = (val_sentences, val_labels),
    callbacks = [
        create_tensorboard_callback(
            dir_name = SAVE_DIR,
            experiment_name = "tf_hub_sentence_encoder"
        )
    ]
)
```

[64] Python

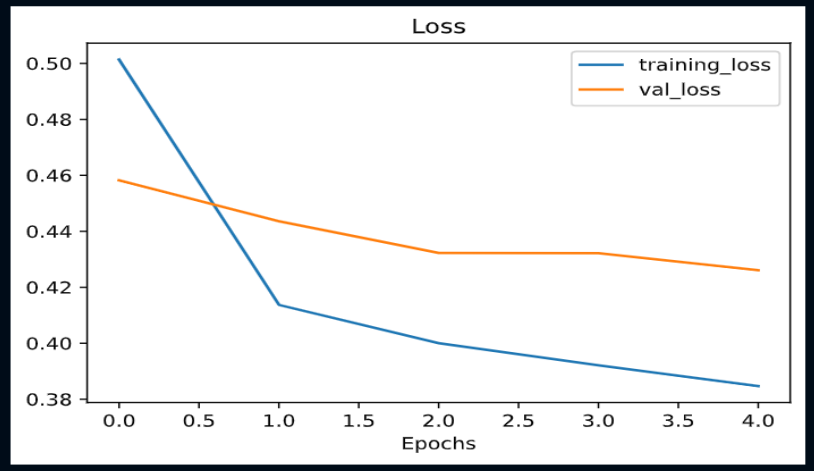
```
Saving TensorBoard log files to: model_logs/tf_hub_sentence_encoder/20210912-141350
Epoch 1/5
```

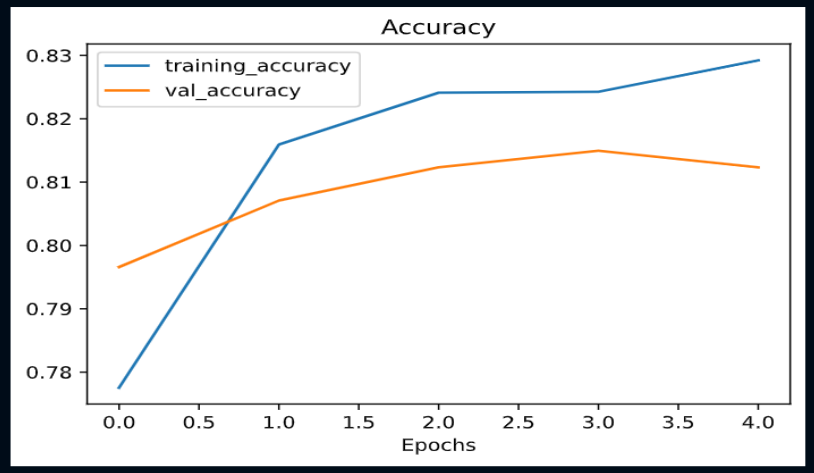
Jupyter Server: local

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
plot_loss_curves(model_6_history)
```

[65] Python





5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
model_6_pred_probs = model_6.predict(val_sentences)
model_6_pred_probs[:10]
```

[66] Python

```
array([[0.19871199],
       [0.74807906],
       [0.9861225 ],
       [0.21762463],
       [0.6936886 ],
       [0.73129445],
       [0.97766495],
       [0.98143315],
       [0.9406043 ],
       [0.1020799 ]], dtype=float32)
```

```
model_6_preds = tf.squeeze(tf.round(model_6_pred_probs))
model_6_preds[:10]
```

[67] Python

```
<tf.Tensor: shape=(10,), dtype=float32, numpy=array([0., 1., 1., 0., 1., 1., 1., 1., 1., 0.], dtype=float32)>
```

```
model_6_results = calculate_results(val_labels, model_6_preds)
model_6_results
```

[68] Python

```
{'accuracy': 81.23359580052494,
 'precision': 0.8140043704592863,
 'recall': 0.8123359580052494,
 'f1': 0.8109868382392855}
```

```
import numpy as np

np.array( list(model_6_results.values())) > np.array( list(baseline_results.values()))
```

[69] Python

```
array([ True,  True,  True,  True])
```

```
# 10% of dataset

train_sentences_90_percent, train_sentences_10_percent, train_labels_90_percent, train_labels_10_percent = train_test_split(
    np.array(train_sentences),
    train_labels,
    test_size = 0.1,
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
print(f"전체 학습 데이터량 : {len(train_sentences)}")
print(f"전체 학습데이터의 10% : {len(train_sentences_10_percent)}")
```

[72] Python

```
전체 학습 데이터량 : 6851
전체 학습데이터의 10% : 686
```

```
pd.Series(train_labels_10_percent).value_counts()
```

[73] Python

```
0    415
1    271
dtype: int64
```

```
model_7 = tf.keras.models.clone_model(model_6)

model_7.compile(
    loss = "binary_crossentropy",
    optimizer = tf.keras.optimizers.Adam(),
    metrics = ["accuracy"]
)

model_7.summary()
```

[74] Python

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
USE (KerasLayer)             (None, 512)               256797824
_________________________________________________________________
dense_5 (Dense)              (None, 64)                32832
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 65
=================================================================
Total params: 256,830,721
Trainable params: 32,897
Non-trainable params: 256,797,824
_________________________________________________________________
```

```
model_7_history = model_7.fit(
    train_sentences_10_percent,
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
model_7_history = model_7.fit(
    train_sentences_10_percent,
    train_labels_10_percent,
    epochs = 5,
    validation_data = (val_sentences, val_labels),
    callbacks = [
        create_tensorboard_callback(
            dir_name = SAVE_DIR,
            experiment_name = "10_percent_tf_hub_sentence_encoder"
        )
    ]
)
```

[75] Python

```
Saving TensorBoard log files to: model_logs/10_percent_tf_hub_sentence_encoder/20210912-145244
Epoch 1/5
22/22 [==============================] - 5s 136ms/step - loss: 0.6700 - accuracy: 0.6676 - val_loss: 0.6540 - val_accuracy: 0.6667
Epoch 2/5
22/22 [==============================] - 0s 13ms/step - loss: 0.6016 - accuracy: 0.8090 - val_loss: 0.5997 - val_accuracy: 0.7402
Epoch 3/5
22/22 [==============================] - 0s 12ms/step - loss: 0.5242 - accuracy: 0.8222 - val_loss: 0.5472 - val_accuracy: 0.7598
Epoch 4/5
22/22 [==============================] - 0s 12ms/step - loss: 0.4587 - accuracy: 0.8367 - val_loss: 0.5102 - val_accuracy: 0.7690
Epoch 5/5
22/22 [==============================] - 0s 13ms/step - loss: 0.4126 - accuracy: 0.8426 - val_loss: 0.4942 - val_accuracy: 0.7769
```

```
model_7_pred_probs = model_7.predict(val_sentences)
model_7_pred_probs[:10]
```

[76] Python

```
array([[0.23759508],
       [0.7631977 ],
       [0.8880272 ],
       [0.32815146],
       [0.524934  ],
       [0.81110156],
       [0.81641316],
       [0.8375648 ],
       [0.80024445],
       [0.12028435]], dtype=float32)
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
model_7_preds = tf.squeeze(tf.round(model_7_pred_probs))
model_7_preds[:10]
```

[79] Python

```
<tf.Tensor: shape=(10,), dtype=float32, numpy=array([0., 1., 1., 0., 1., 1., 1., 1., 1., 0.], dtype=float32)>
```

```
model_7_results = calculate_results(val_labels, model_7_preds)
model_7_results
```

[80] Python

```
{'accuracy': 77.69028871391076,
 'precision': 0.7822241302284023,
 'recall': 0.7769028871391076,
 'f1': 0.7734519762210931}
```

```
np.array( list(model_7_results.values())) > np.array( list(baseline_results.values()))
```

[81] Python

```
array([False, False, False, False])
```

```
all_model_results = pd.DataFrame({
    "baseline" : baseline_results,
    "simple_dense" : model_1_results,
    "lstm" : model_2_results,
    "gru" : model_3_results,
    "bidiretional" : model_4_results,
    "conv1d" : model_5_results,
    "tf_hub_sentence_encoder" : model_6_results,
    "tf_hub_10_percent_data" : model_7_results,
})
all_model_results = all_model_results.transpose()
all_model_results
```

[143] Python

	accuracy	precision	recall	f1
baseline	79.265092	0.811139	0.792651	0.786219
simple_dense	78.608924	0.790739	0.786089	0.783154
lstm	78.215223	0.782913	0.782152	0.780749
gru	77.034121	0.771825	0.770341	0.768323
bidiretional	76.246719	0.762124	0.762467	0.761710
conv1d	75.984252	0.760134	0.759843	0.758354
tf_hub_sentence_encoder	81.233596	0.814004	0.812336	0.810987
tf_hub_10_percent_data	77.690289	0.782224	0.776903	0.773452

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

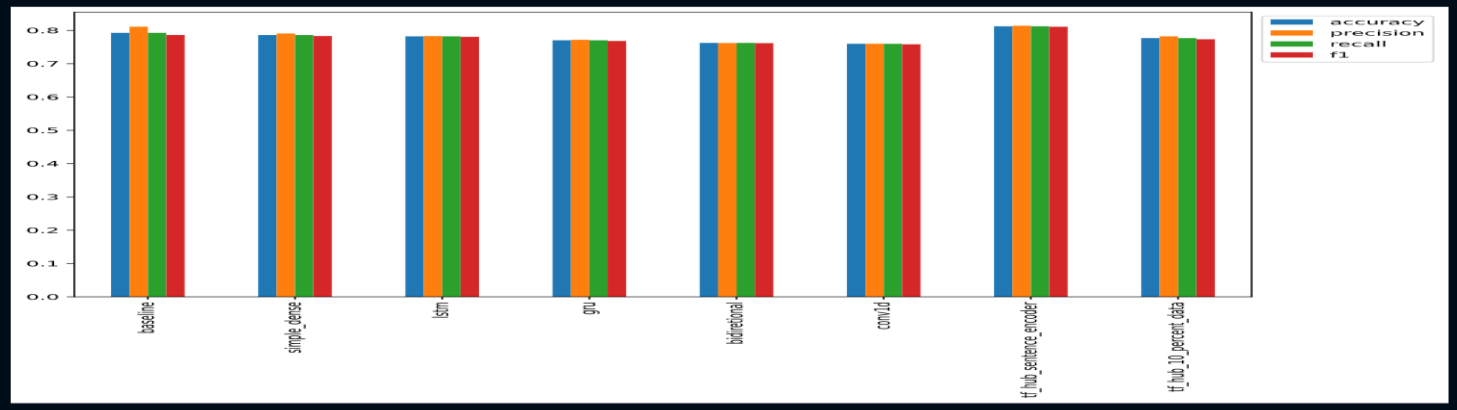
```
[144]
all_model_results["accuracy"] = all_model_results["accuracy"] / 100
```
Python

```
[145]
all_model_results
```
Python

	accuracy	precision	recall	f1
baseline	0.792651	0.811139	0.792651	0.786219
simple_dense	0.786089	0.790739	0.786089	0.783154
lstm	0.782152	0.782913	0.782152	0.780749
gru	0.770341	0.771825	0.770341	0.768323
bidiretional	0.762467	0.762124	0.762467	0.761710
conv1d	0.759843	0.760134	0.759843	0.758354
tf_hub_sentence_encoder	0.812336	0.814004	0.812336	0.810987
tf_hub_10_percent_data	0.776903	0.782224	0.776903	0.773452

```
[147]
all_model_results.plot(kind = "bar", figsize = (10, 7)).legend(bbox_to_anchor=(1.0, 1.0))
```
Python

```
<matplotlib.legend.Legend at 0x21d3afc39d0>
```



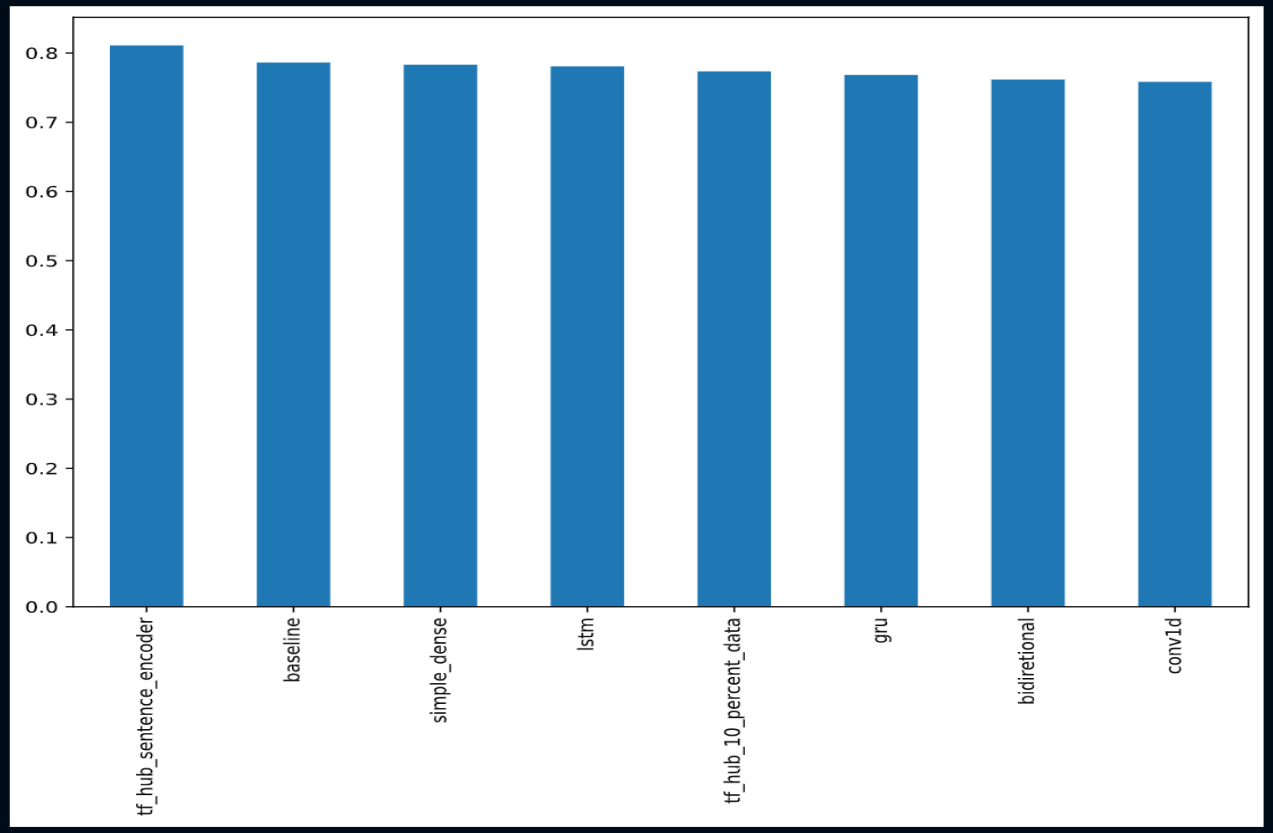

```
all_model_results.sort_values("f1", ascending = False)["f1"].plot(kind = "bar", figsize=(10, 7))
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```
[148]  all_model_results.sort_values("f1", ascending = False)["f1"].plot(kind = "bar", figsize=(10, 7))
```

Python

```
...  <AxesSubplot:>
```




```
baseline_pred_probs = np.max(model_0.predict_proba(val_sentences), axis = 1)
combined_pred_probs = baseline_pred_probs + tf.squeeze(model_2_pred_probs, axis = 1) + tf.squeeze(model_6_pred_probs)
combined_preds = tf.round(combined_pred_probs / 3)
```

5~8주차 : Tensorflow 기초, Tensorflow로 하는 자연어 처리

```python
baseline_pred_probs = np.max(model_0.predict_proba(val_sentences), axis = 1)
combined_pred_probs = baseline_pred_probs + tf.squeeze(model_2_pred_probs, axis = 1) + tf.squeeze(model_6_pred_probs)
combined_preds = tf.round(combined_pred_probs / 3)
combined_preds[:20]
```

[150] Python

```
<tf.Tensor: shape=(20,), dtype=float32, numpy=
array([0., 1., 1., 0., 0., 1., 1., 1., 1., 0., 0., 1., 0., 1., 0., 0., 0.,
       0., 0., 1.], dtype=float32)>
```

```python
ensenble_results = calculate_results(val_labels, combined_preds)
ensenble_results
```

[151] Python

```
{'accuracy': 79.52755905511812,
 'precision': 0.7950188004299005,
 'recall': 0.7952755905511811,
 'f1': 0.7948762980229479}
```

```python
all_model_results.loc["ensemble_results"] = ensenble_results
```

[153] Python

```python
all_model_results.loc["ensemble_results"]["accuracy"] = all_model_results.loc["ensemble_results"]["accuracy"] / 100
```

[155] Python

```python
all_model_results
```

[156] Python

	accuracy	precision	recall	f1
baseline	0.792651	0.811139	0.792651	0.786219
simple_dense	0.786089	0.790739	0.786089	0.783154
lstm	0.782152	0.782913	0.782152	0.780749
gru	0.770341	0.771825	0.770341	0.768323
bidiretional	0.762467	0.762124	0.762467	0.761710
conv1d	0.759843	0.760134	0.759843	0.758354
tf_hub_sentence_encoder	0.812336	0.814004	0.812336	0.810987
tf_hub_10_percent_data	0.776903	0.782224	0.776903	0.773452
ensemble_results	0.795276	0.795019	0.795276	0.794876

Python을 이용한 이미지 편집 어플 만들기 (Streamlit이용)

Python을 이용한 어플 만들기(Streamlit이용)

streamlit_app_002.py | streamlit_app_004.py | streamlit_app_001.py | streamlit_app_003.py

C: > Users > 98040 > Downloads > streamlit_app_001.py > ...

```python
# streamlit_app_001.py
import streamlit as st

def main():
    st.title("이미지 편집 앱")
    st.text("빠르고 쉽게 이미지를 편집")
    st.text("아주 쉽습니다!")

if __name__ == "__main__":
    main()
```

Browser링크 생성

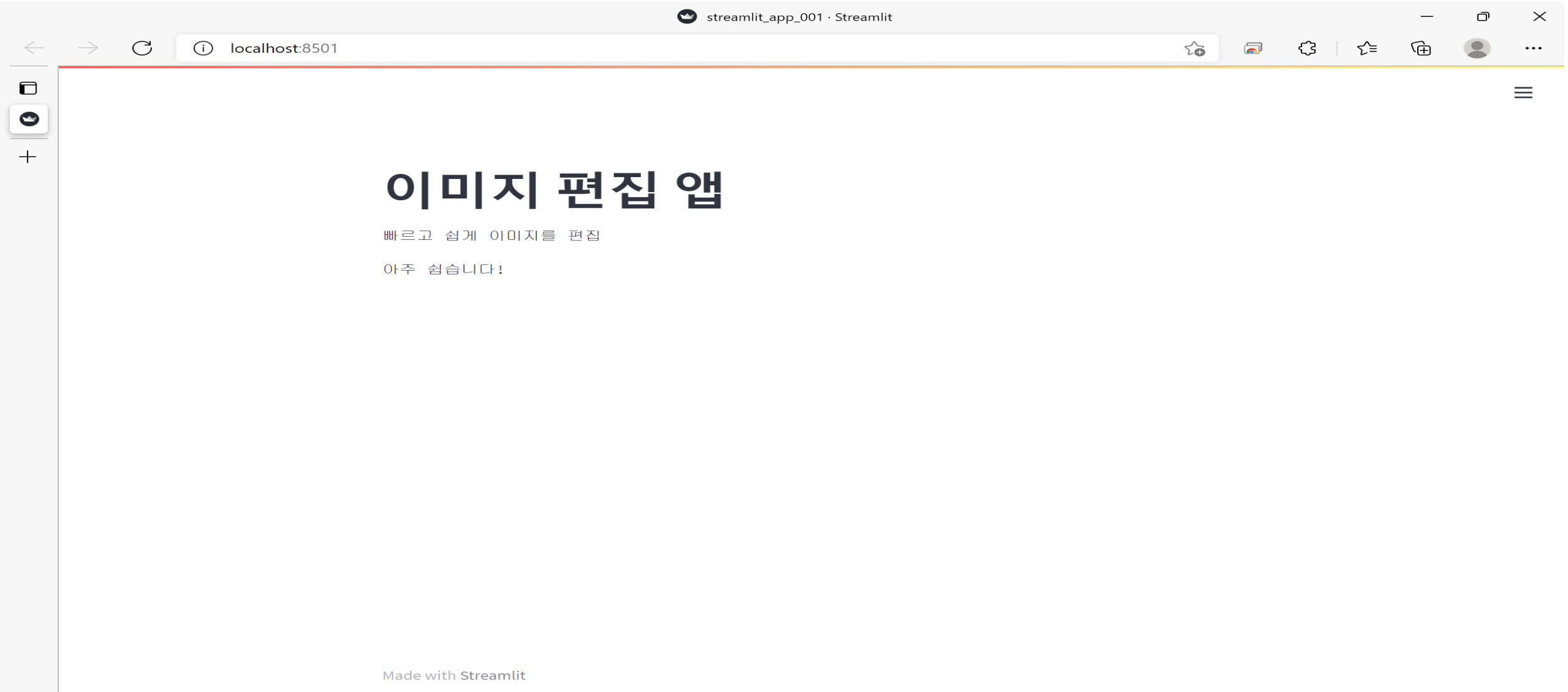
```
You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.219.104:8501
```

문제 출력 터미널 디버그 콘솔

```
loads/streamlit_app_001.py
2021-11-10 21:32:03.014
  Warning: to view this Streamlit app on a browser, run it with the following
  command:

    streamlit run c:/Users/98040/Downloads/streamlit_app_001.py [ARGUMENTS]
PS C:\Users\98040> & C:/Users/98040/AppData/Local/Programs/Python/Python38/python.exe c:/Users/98040/Downloads/streamlit_app_001.py
2021-11-10 21:32:26.204
  Warning: to view this Streamlit app on a browser, run it with the following
  command:

    streamlit run c:/Users/98040/Downloads/streamlit_app_001.py [ARGUMENTS]
PS C:\Users\98040> streamlit run c:/Users/98040/Downloads/streamlit_app_001.py
```

줄 10, 열 11 공백: 4 UTF-8

Python을 이용한 어플 만들기(Streamlit이용)



Python을 이용한 어플 만들기(Streamlit이용)

streamlit_app_002.py | streamlit_app_004.py | streamlit_app_001.py | streamlit_app_003.py

C: > Users > 98040 > Downloads > streamlit_app_002.py > main

```python
# streamlit_app_002.py
import streamlit as st
from PIL import Image, ImageEnhance

def main():
    st.title("이미지 편집 앱")
    st.text("빠르고 쉽게 이미지를 편집")
    st.text("아주 쉽습니다!")

    activities = ["Detection", "About"]
    choice = st.sidebar.selectbox("할 일을 선택해주세요", activities)

    if choice == "Detection":
        st.subheader("얼굴 인식")
        image_file = st.file_uploader("이미지 업로드", type=["jpg", "png", "jpeg"])

        if image_file is not None:
            our_image = Image.open(image_file)
            st.text("원본 이미지")
            st.image(our_image)

if __name__ == "__main__":
    main()
```

문제 출력 터미널 디버그 콘솔 Python

```
PS C:\Users\98040> & C:/Users/98040/AppData/Local/Programs/Python/Python38/python.exe c:/Users/98040/Downloads/streamlit_app_002.py
2021-11-10 21:36:30.232
  Warning: to view this Streamlit app on a browser, run it with the following
  command:

    streamlit run c:/Users/98040/Downloads/streamlit_app_002.py [ARGUMENTS]
PS C:\Users\98040>  streamlit run c:/Users/98040/Downloads/streamlit_app_002.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.219.104:8501
```

Python을 이용한 어플 만들기(Streamlit이용)

할 일을 선택해주세요

Detection

이미지 편집 앱

빠르고 쉽게 이미지를 편집

아주 쉽습니다!

얼굴 인식

이미지 업로드

Drag and drop file here

Limit 200MB per file • JPG, PNG, JPEG

Browse files

전소민.jpg 78.8KB

원본 이미지

Python을 이용한 어플 만들기(Streamlit이용)

streamlit_app_002.py | streamlit_app_004.py | streamlit_app_001.py | streamlit_app_003.py

C: > Users > 98040 > Downloads > streamlit_app_003.py > ...

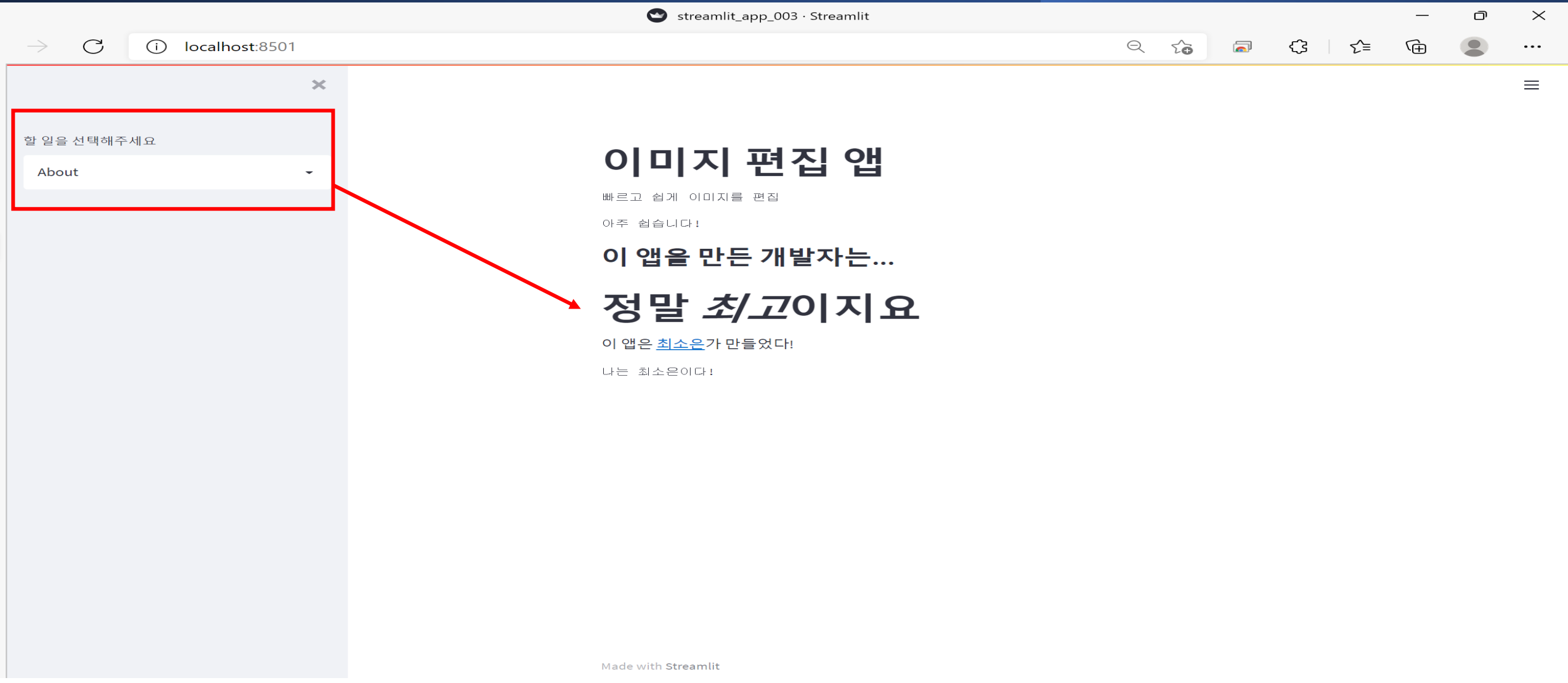
```python
# streamlit_app_003.py
import streamlit as st
from PIL import Image, ImageEnhance

def main():
    st.title("이미지 편집 앱")
    st.text("빠르고 쉽게 이미지를 편집")
    st.text("아주 쉽습니다!")

    activities = ["Detection", "About"]
    choice = st.sidebar.selectbox("할 일을 선택해주세요", activities)

    if choice == "Detection":
        st.subheader("얼굴 인식")
        image_file = st.file_uploader("이미지 업로드", type=["jpg", "png", "jpeg"])

        if image_file is not None:
            our_image = Image.open(image_file)
            st.text("원본 이미지")
            st.image(our_image)
    elif choice == "About":
        st.subheader("이 앱을 만든 개발자는...")
        st.markdown("# 정말 _최고_이지요")
        st.markdown("이 앱은 [최소은](http://www.naver.com)가 만들었다!")
        st.text("나는 최소은이다!")


if __name__ == "__main__":
    main()
```

문제 출력 터미널 디버그 콘솔 Python

```
PS C:\Users\98040> & C:/Users/98040/AppData/Local/Programs/Python/Python38/python.exe c:/Users/98040/Downloads/streamlit_app_003.py
2021-11-10 21:39:52.423
  Warning: to view this Streamlit app on a browser, run it with the following
  command:

    streamlit run c:/Users/98040/Downloads/streamlit_app_003.py [ARGUMENTS]
PS C:\Users\98040>  streamlit run c:/Users/98040/Downloads/streamlit_app_003.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.219.104:8501
```

줄 29, 열 11 공백: 4 UTF-8 CRLF Python

Python을 이용한 어플 만들기(Streamlit이용)



Python을 이용한 어플 만들기(Streamlit이용)

streamlit_app_002.py | streamlit_app_004.py | streamlit_app_001.py | streamlit_app_003.py

C: > Users > 98040 > Downloads > streamlit_app_004.py > ...

```python
# streamlit_app_004.py
import streamlit as st
import cv2
import numpy as np
from PIL import Image, ImageEnhance


def main():
    st.title("이미지 편집 앱")
    st.text("빠르고 쉽게 이미지를 편집")
    st.text("아주 쉽습니다!")

    activities = ["Detection", "About"]
    choice = st.sidebar.selectbox("할 일을 선택해주세요", activities)

    if choice == "Detection":
        st.subheader("얼굴 인식")
        image_file = st.file_uploader("이미지 업로드", type=["jpg", "png", "jpeg"])

        if image_file is not None:
            our_image = Image.open(image_file)
            st.text("원본 이미지")
            # st.image(our_image)

            enhance_list = ["원본", "Gray-scale", "Contrast", "Brightness", "Blurring", "Sharpness"]
            enhance_type = st.sidebar.radio("처리방법", enhance_list)

            if enhance_type == "Gray-scale":
                img = np.array(our_image.convert("RGB"))
                gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
                st.image(gray)
            elif enhance_type == "Contrast":
                rate = st.sidebar.slider("Contrast", 0.5, 6.0)
                enhancer = ImageEnhance.Contrast(our_image)
                enhanced_img = enhancer.enhance(rate)
                st.image(enhanced_img)
```

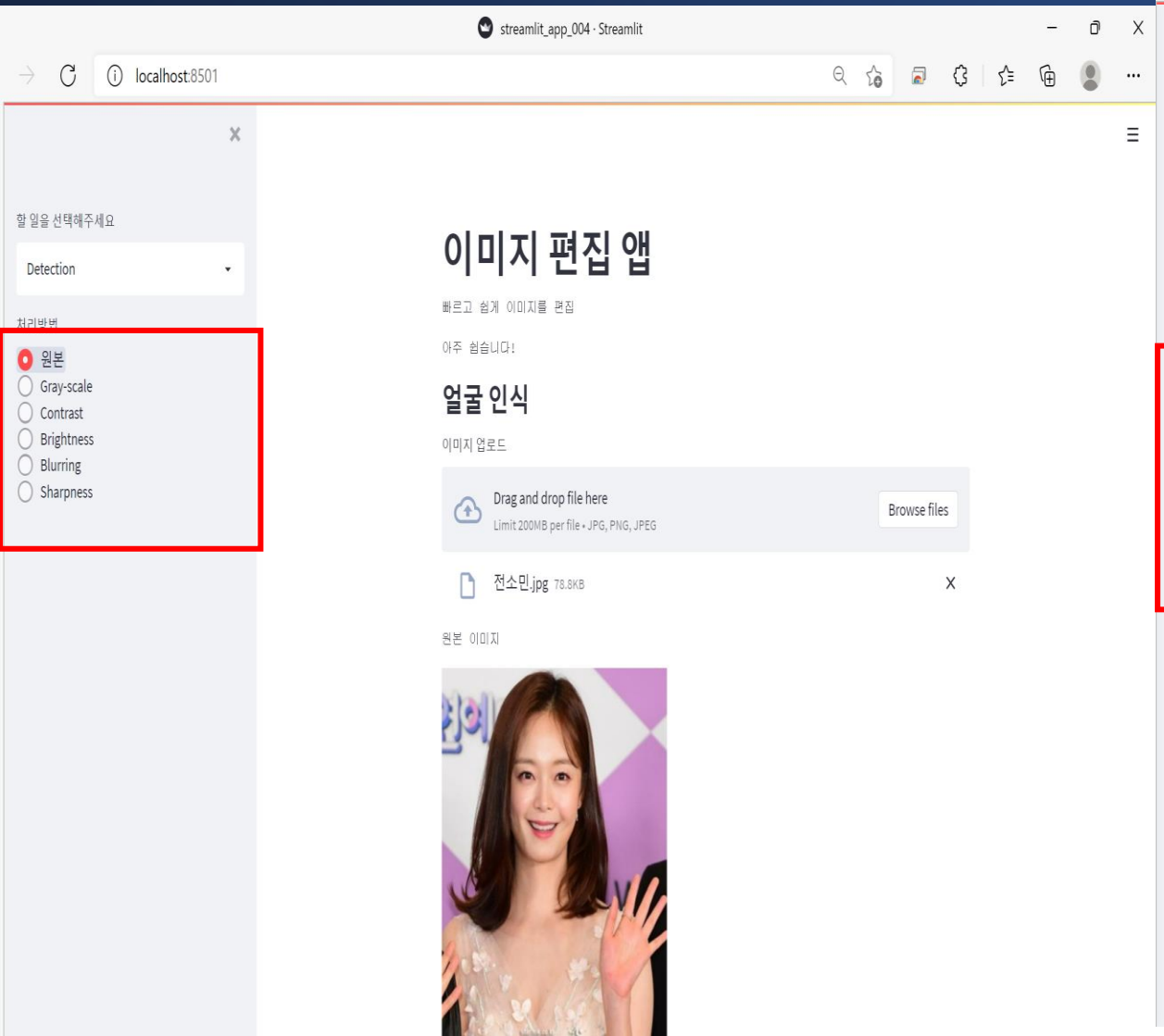
Python을 이용한 어플 만들기(Streamlit이용)

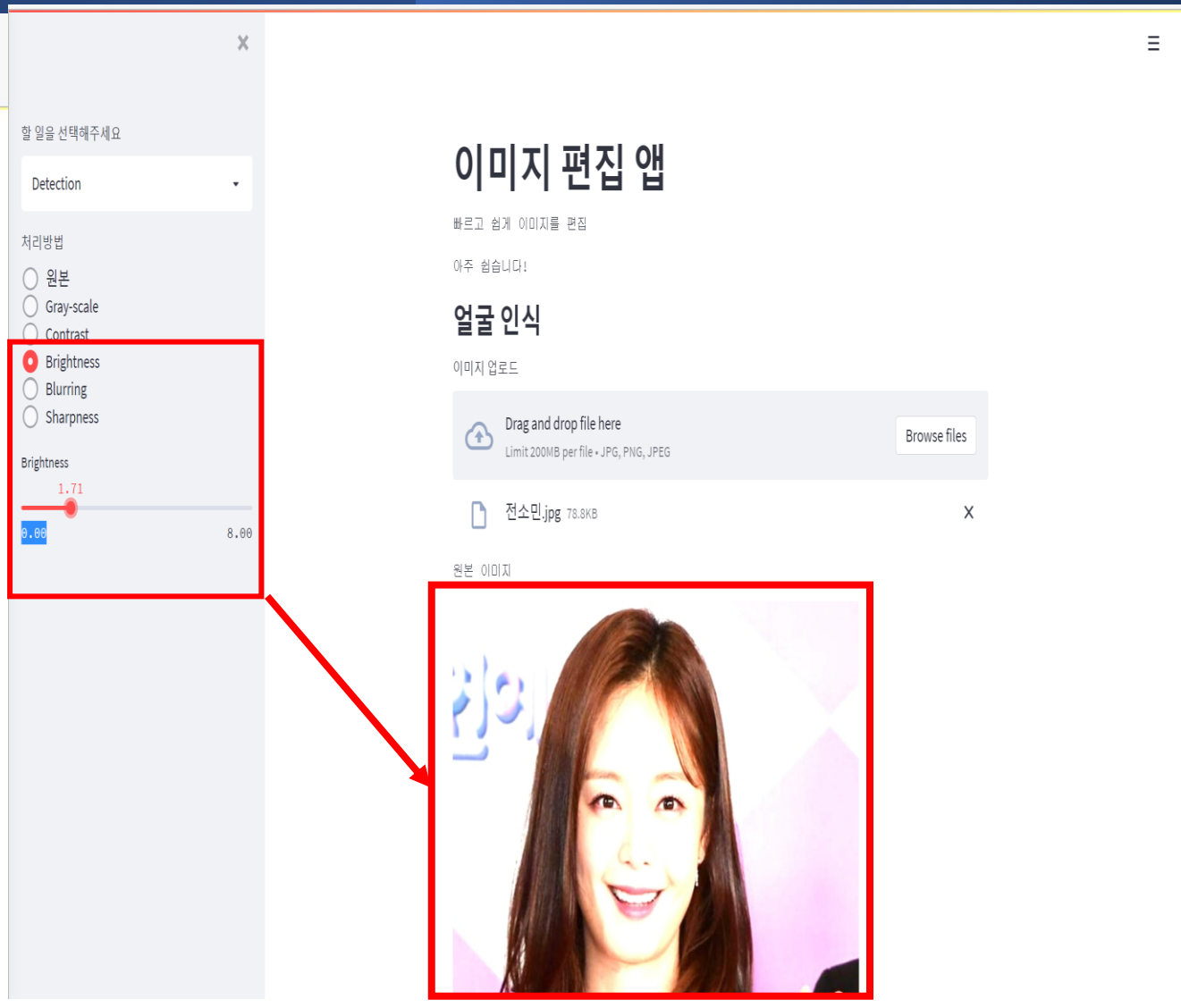
```
            elif enhance_type == "Brightness":
                rate = st.sidebar.slider("Brightness", 0.0, 8.0)
                enhancer = ImageEnhance.Brightness(our_image)
                enhanced_img = enhancer.enhance(rate)
                st.image(enhanced_img)
            elif enhance_type == "Blurring":
                rate = st.sidebar.slider("Blurring", 0.0, 7.0)
                burred_img = cv2.GaussianBlur(np.array(our_image), (15, 15), rate)
                st.image(burred_img)
            elif enhance_type == "Sharpness":
                rate = st.sidebar.slider("Sharpness", 0.0, 14.0)
                enhancer = ImageEnhance.Sharpness(our_image)
                enhanced_img = enhancer.enhance(rate)
                st.image(enhanced_img)
            elif enhance_type == "원본":
                st.image(our_image, width = 300)
            else:
                st.image(our_image, width = 300)

    elif choice == "About":
        st.subheader("이 앱을 만든 개발자는...")
        st.markdown("# 정말 _최고_이지요")
        st.markdown("이 앱은 [최소은](http://www.naver.com)가 만들었다!")
        st.text("나는 최소은이다!")


if __name__ == "__main__":
    main()
```

문제 출력 터미널 디버그 콘솔 Python

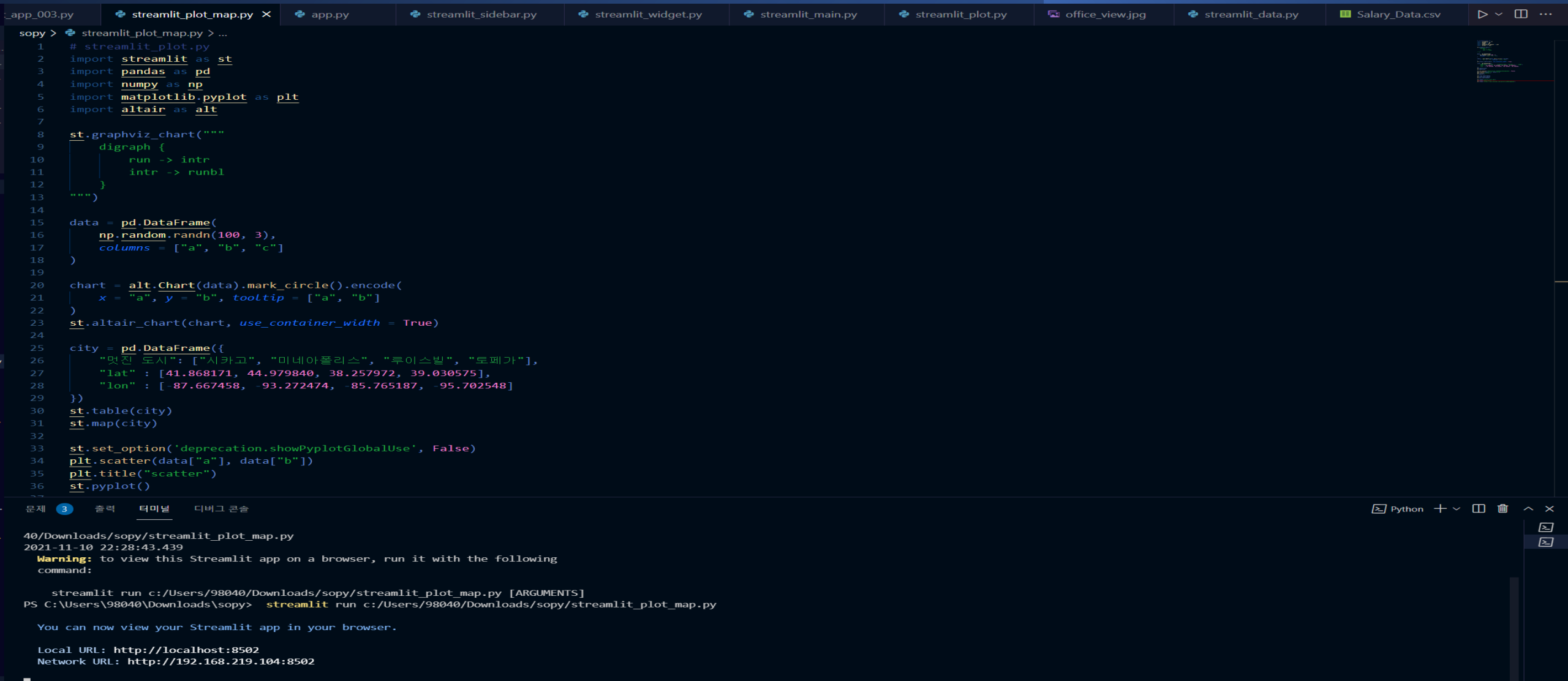
```
2021-11-10 21:42:51.242
  Warning: to view this Streamlit app on a browser, run it with the following
  command:

    streamlit run c:/Users/98040/Downloads/streamlit_app_004.py [ARGUMENTS]
PS C:\Users\98040>
PS C:\Users\98040>     streamlit run c:/Users/98040/Downloads/streamlit_app_004.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.219.104:8501
```

Python을 이용한 어플 만들기(Streamlit이용) 결과

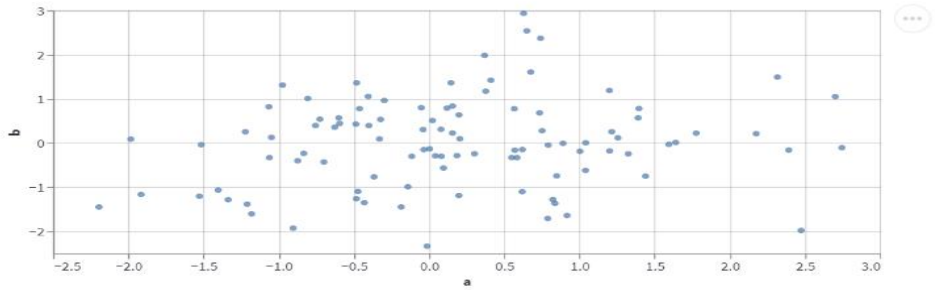




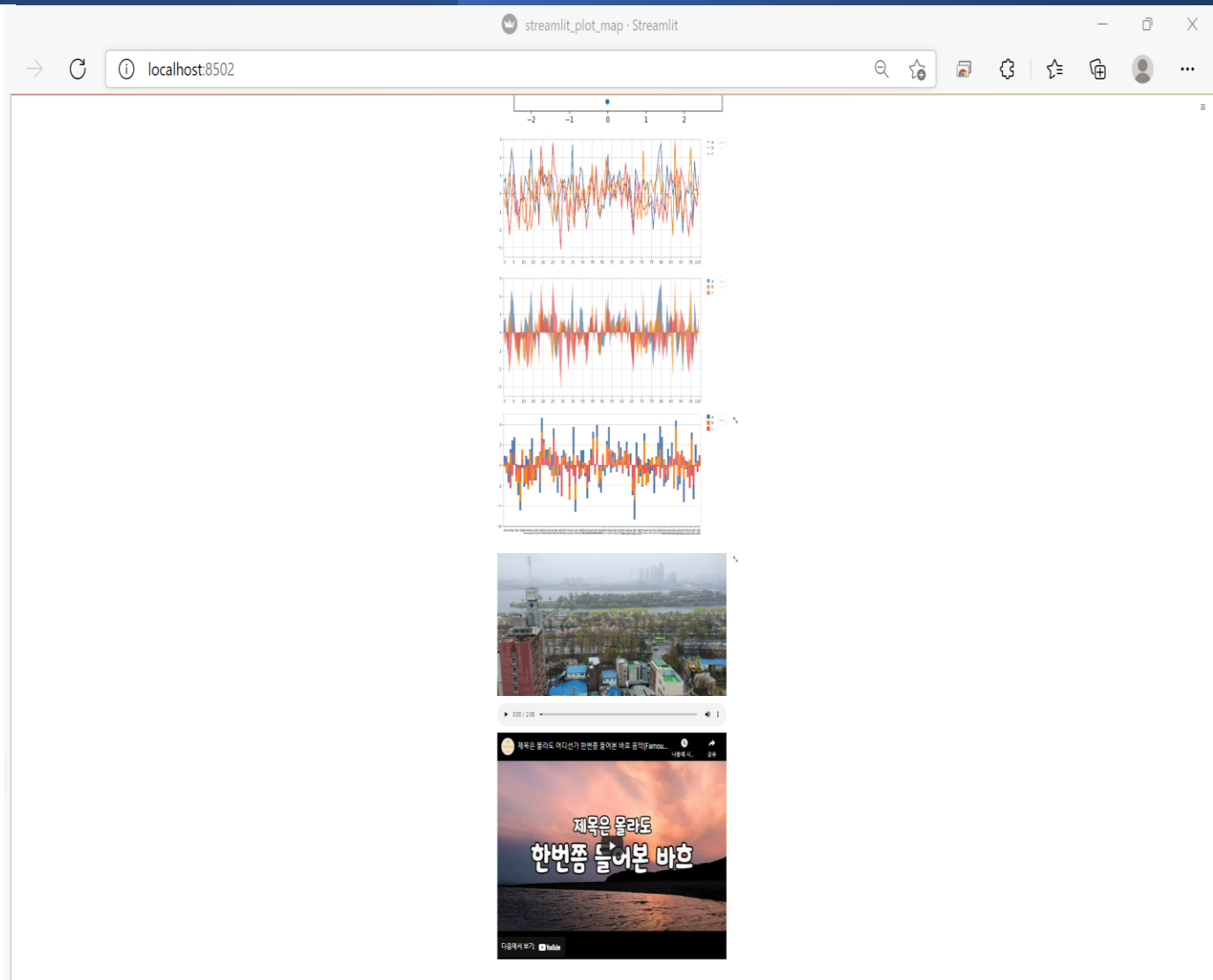
Python을 이용한 어플 만들기(Streamlit이용) 결과



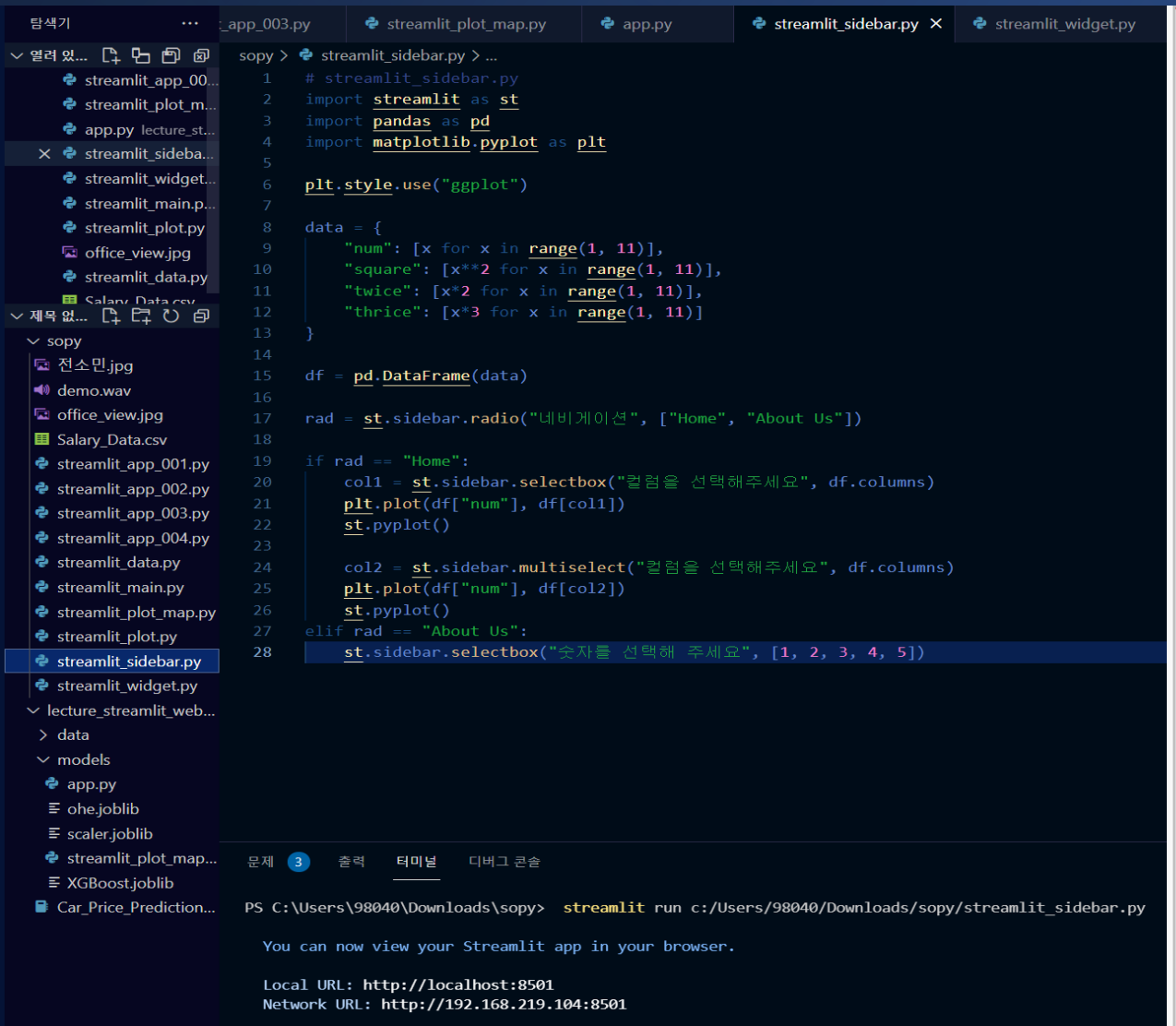
Python을 이용한 어플 만들기(Streamlit이용) 결과

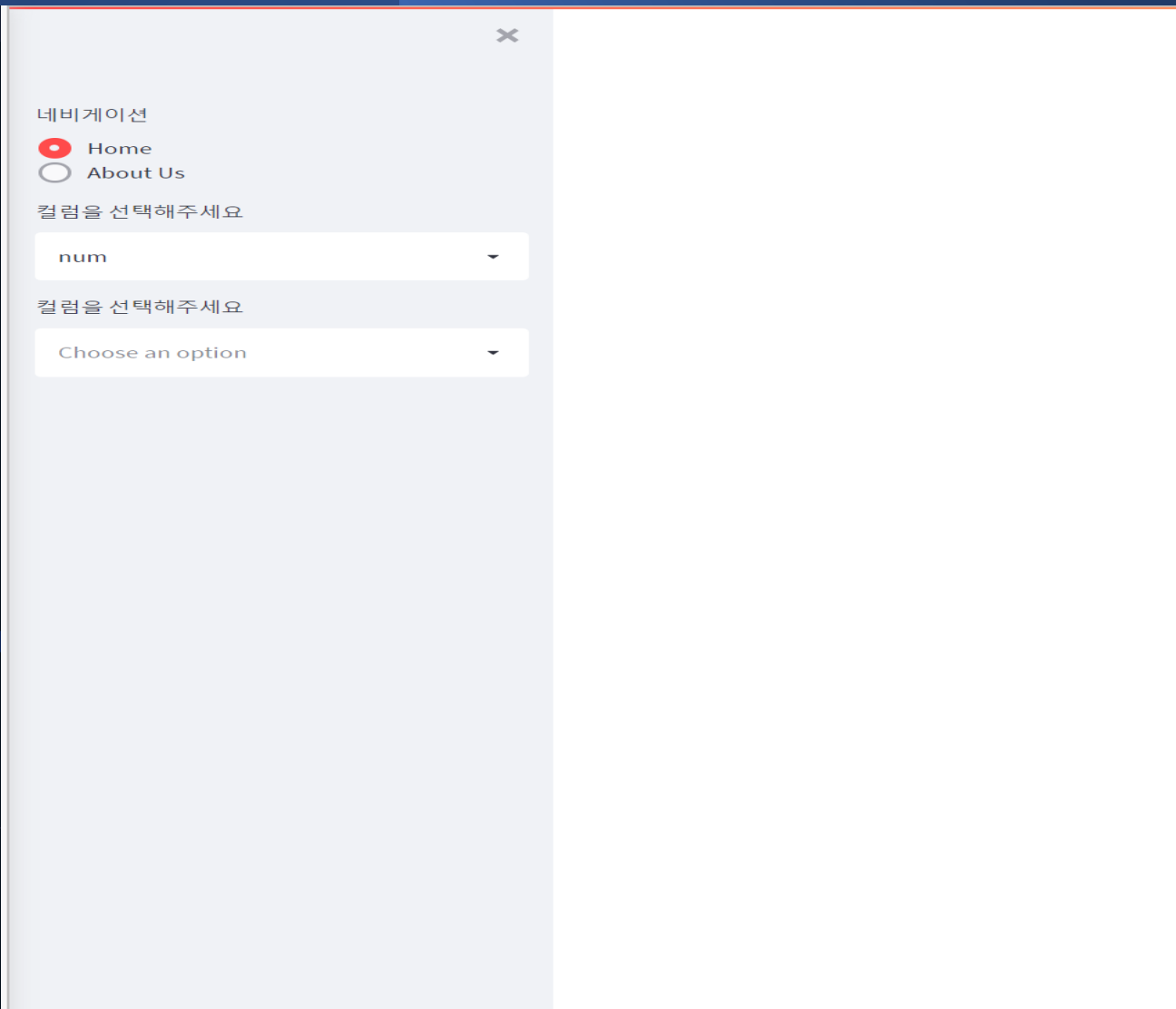


	멋진 도시	lat	lon
0	시카고	41.8682	-87.6675
1	미네아폴리스	44.9798	-93.2725
2	루이스빌	38.2580	-85.7652
3	토페가	39.0306	-95.7025



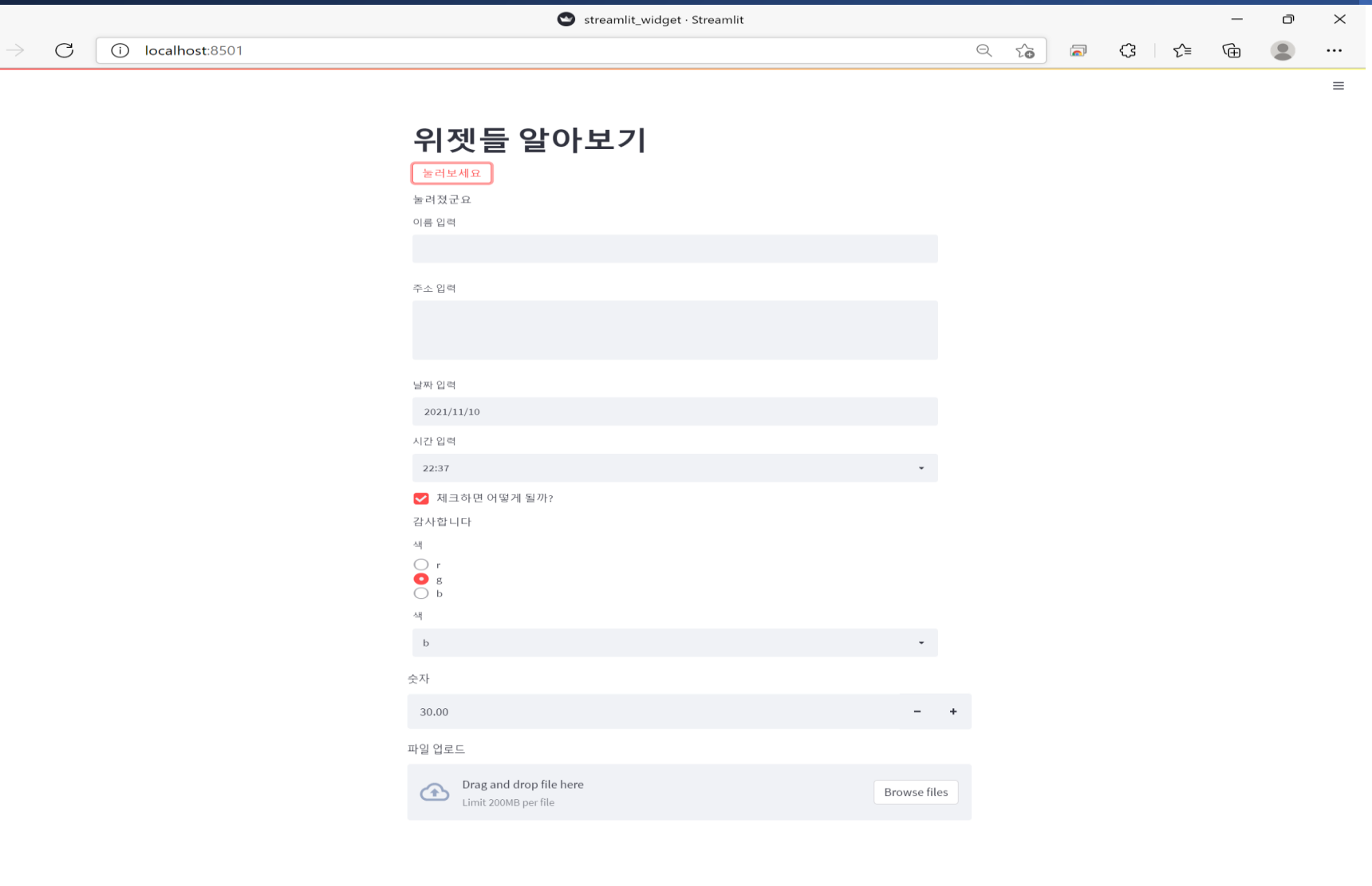
Python을 이용한 어플 만들기(Streamlit이용) 결과



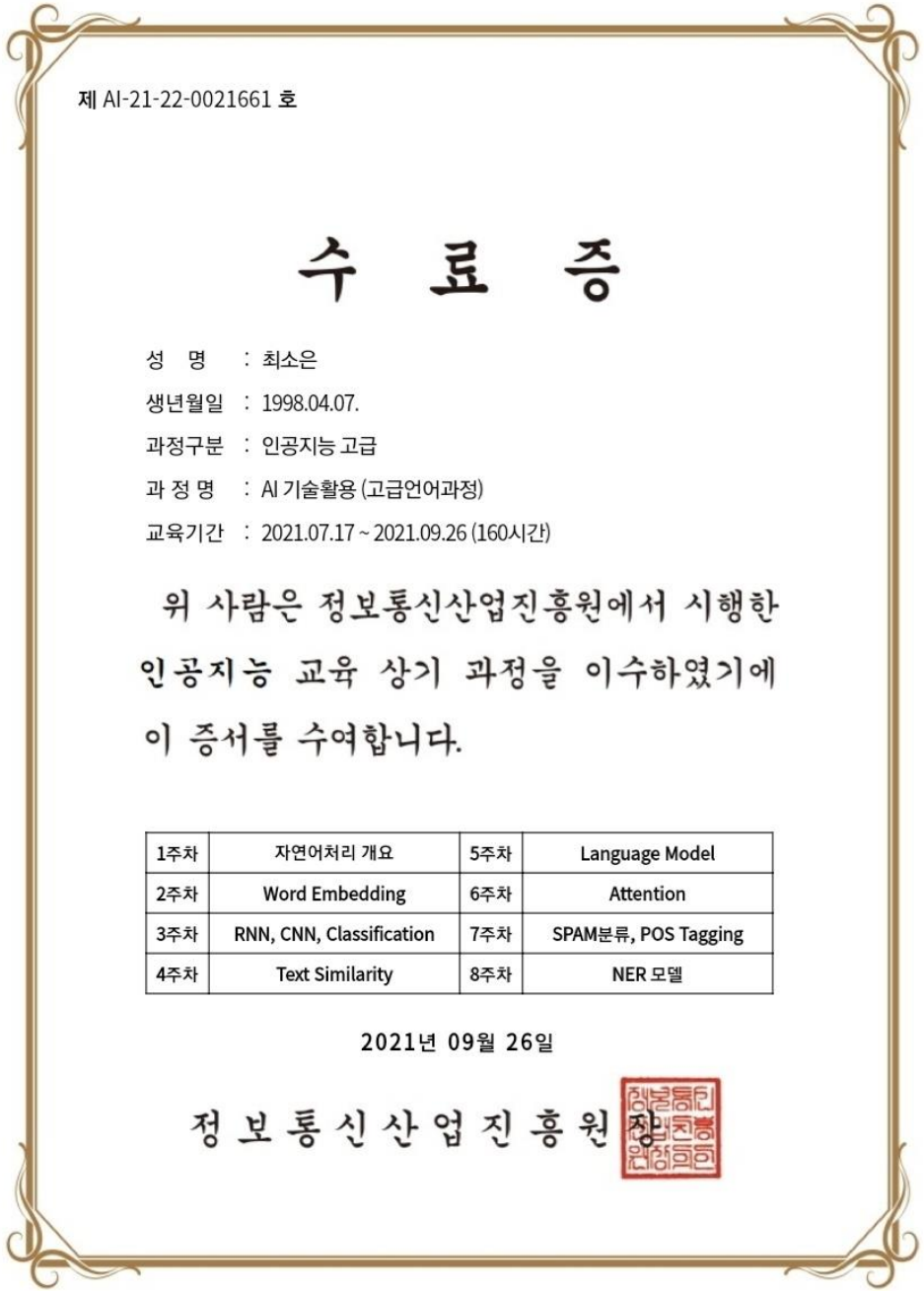


Python을 이용한 어플 만들기(Streamlit이용) 결과

Python을 이용한 어플 만들기(Streamlit이용) 결과



인공지능 고급 언어지능 프로젝트 개발 수료증



제 AI-21-22-0021661 호

수 료 증

성 명 : 최소은

생년월일 : 1998.04.07.

과정구분 : 인공지능 고급

과 정 명 : AI 기술활용 (고급언어과정)

교육기간 : 2021.07.17 ~ 2021.09.26 (160시간)

위 사람은 정보통신산업진흥원에서 시행한 인공지능 교육 상기 과정을 이수하였기에 이 증서를 수여합니다.

1주차	자연어처리 개요	5주차	Language Model
2주차	Word Embedding	6주차	Attention
3주차	RNN, CNN, Classification	7주차	SPAM분류, POS Tagging
4주차	Text Similarity	8주차	NER 모델

2021년 09월 26일

정보통신산업진흥원장