

Part 7. Flask 기반 감성분석 API 개발



7. Flask 기반 감성분석 API 개발

네이버 영화리뷰 감성분석 개요
감성분석 API 개발 방향
API 정의
Add Deep Learning model handler
Unittest
Flask API 개발 & 배포
Test API on remote

네이버 영화리뷰는 자연어 처리 분야에서 기본적인 benchmark dataset으로 많이 활용되고 있음

네이버 영화리뷰 감성분석 개요

네이버 영화리뷰 데이터로 학습한 ML/DL 모델을 활용해 감성분석 API 개발

나이브베이즈 모델과 딥러닝 모델로 학습한 두 개의 모델을 서빙하며 0은 부정성 1은 긍정성을 의미
학습에 사용한 데이터는 박은정님이 공개한 NSMC 데이터

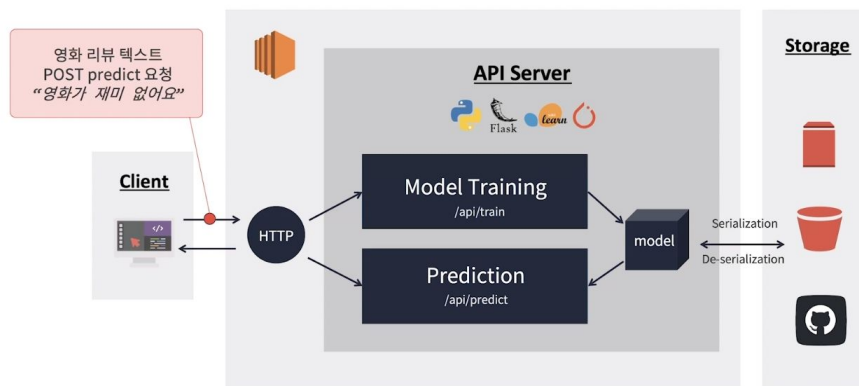
id	document	label
9976970	아 더빙.. 진짜 짜증나네요 목소리	0
3819312	흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나	1
10265843	너무재밌었다그래서보는것을추천한다	0
9045019	교도소 이야기구먼 ..솔직히 재미는 없다..평점 조정	0
6483659	사이몬페그의 익살스런 연기가 돋보였던 영화!스파이더맨에서 늙어보이기만 했던 커스틴 ...	1
5403919	막 걸음마 떤 3세부터 초등학교 1학년생인 8살용영화.ㅋㅋㅋ...별반개도 아까움.	0
7797314	원작의 긴장감을 제대로 살려내지못했다.	0

Source: <https://github.com/efkthmc/>

감성분석 API는 먼저 클라이언트에서 POST메소드로 predict를 해달라고 요청을 할 것이고
요청할때 text가 "영화가 재미 없어요"라고 보냈을 때 model_type을 함께 인자로 전달해서

감성분석 API 개발 방향

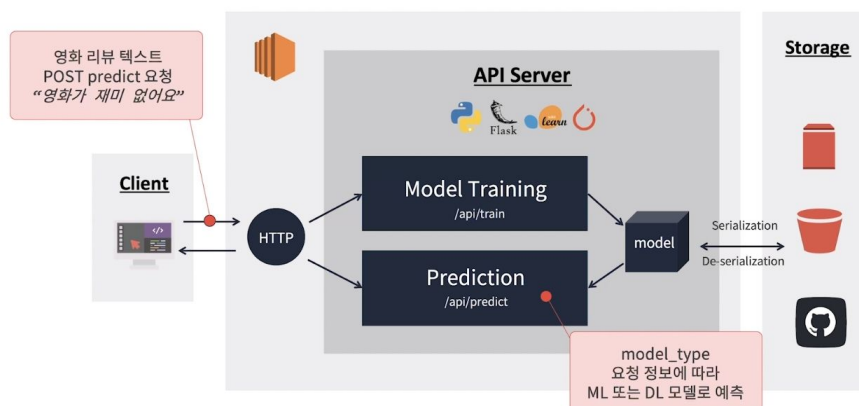
AWS EC2와 Python Flask 기반 모델 학습 및 추론을 요청/응답하는 API 서버 개발



요청정보에 따라서 머신러닝 모델 또는 딥러닝 모델로 예측을 하는 API를 개발할 것이다.

감성분석 API 개발 방향

AWS EC2와 Python Flask 기반 모델 학습 및 추론을 요청/응답하는 API 서버 개발

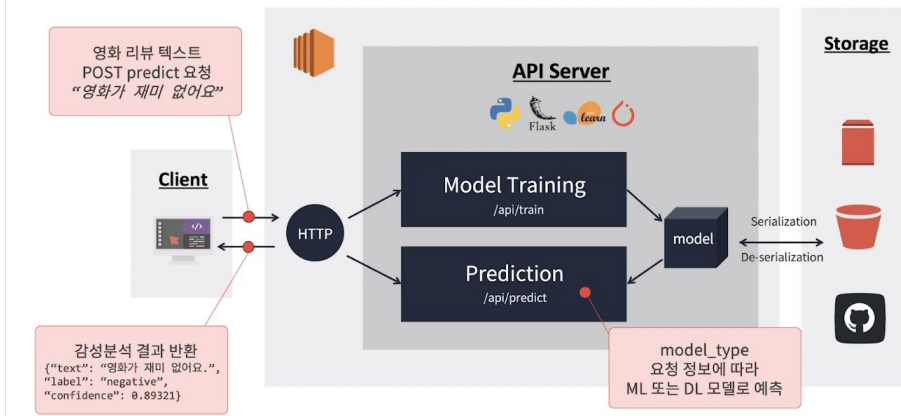


예측한 결과에 대해서 레이블을 달아주고 얼마만큼 확률 값이 나오는지도 최종적으로 클라이언트에게 결과를 알려주려고 한다.

우측에서 사전에 학습된 데이터는 EC2와 연결된 EBS에 사전에 학습했던 머신러닝 모델과 외부 Repository에 저장된 딥러닝 모델 2가지를 사용할 것이다.

감성분석 API 개발 방향

AWS EC2와 Python Flask 기반 모델 학습 및 추론을 요청/응답하는 API 서버 개발



RESTFUL API에서는 서버나 클라이언트 간에 통신을 위해서 인터페이스를 정해야 하는데 이 실습에서는 key, value 형태의 json 포맷으로 요청을 받을 수 있도록 하고 텍스트의 개수에 따라서 인덱스 별로 key와 value로 결과를 저장한 json 포맷으로 최종 output을 반환 할 것이다.

우측에 sample을 봤을때 json형태로 request할때 데이터를 다음과 같이 명시를 해서 요청을 해야 하고 그 결과로서는 아래와 같은 format으로 결과를 반환을 해 줄 것이다.

API 정의

key: value 형태의 json 포맷으로 요청을 받아 text index 별로 key: value로 결과를 저장한 json 포맷으로 결과 반환

POST 방식으로 data를 인자로 하여 predict 요청

text는 review 텍스트들의 리스트

`model_type`를 "ml"로 할 경우, 머신러닝 모델로 추론하도록 함
"dl"의 경우, 딥러닝 모델로 추론하도록 함

```
## Request Data
json = {
    "text": ["review1", "review2", ..., ],
    "model_type": "ml", # or "dl"
}

## Response
# {
#   "idx_0": {
#     "text": "review1",
#     "label": "positive or 'negative'",
#     "confidence": float
#   },
#   "idx_n": {...},
#   ...
# }
```

이제는 모델 타입에 따라서 예측할 수있도록 개발할 것이다.

Add Deep Learning model handler (1/2)

사전 학습한 딥러닝 모델을 활용하여 머신러닝 모델 handler와 동일한 입력에 대해 동일한 결과를 반환하는 handler 개발
사전 학습한 모델은 Hugging Face에서 제공하는 외부 저장소에서 다운로드 받아 불러옴
GPU를 사용할 수 있는 환경인 경우 cuda:0을, false인 경우 cpu를 사용하도록 설정

```
import torch

class DLModelHandler(ModelHandler):
    def __init__(self):
        super().__init__()
        self.initialize()

    def initialize(self):
        # Loading transformer and Re-serializing model
        from transformers import AutoTokenizer, AutoModelForSequenceClassification
        from transformers.file_utils import is_torch_cuda_available
        self.model_name_or_path = 'sakoch/bert-base-multilingual-cased-nmc'
        self.tokenizer = AutoTokenizer.from_pretrained(self.model_name_or_path)
        self.model = AutoModelForSequenceClassification.from_pretrained(self.model_name_or_path)

        # if cuda is available, use GPU device
        self.device = 'cuda:0' if is_torch_cuda_available else 'cpu'
        self.model.to(self.device)
```

딥러닝 모델 같은 경우 initialize하는 방식이 머신러닝과 약간 다르다. 여기서 우리는 transformer라는 라이브러리를 활용할 것이다. 자연어 처리에서 굉장히 많이 사용되고 있고 최근에는 cv나 stt같은 음성 분야, 정형데이터 등에서 사용되고있는 대표적인 인공지능 프레임 워크중에 하나이다. 이 transformer 라이브러리를 활용해서 사전학습된 모델을 불러오고 앞에 ML모델에서 했던 것 처럼 vectorizing 작업을 하는 tokenizer 2개를 사용할 것이다. 모델은 감성분석 데이터로 학습한 모델을 사용할 예정이다. 딥러닝 모델은 주로 gpu환경에서 작동하도록 구성하기 때문에 gpu사용할 수있는 환경을 체크하고 사용할 수없다면 cpu를 사용하도록 설정한다. 반드시 모델 input 데이터들을 gpu에서 처리하는 경우에는 동일한 메모리 환경에서 연산 할 수있도록 gpu를 사용할 때는 gpu의 데이터와 모델을 불러오고 cpu환경에서는 일반 메모리에 올리면 된다.

to()메서드를 이용해서 설정한 디바이스로 모델을 옮겨준다

Add Deep Learning model handler (2/2)

사전 학습한 딥러닝 모델을 활용하여 머신러닝 모델 handler와 동일한 입력에 대해 동일한 결과를 반환하는 handler 개발

```
# ...

def preprocess(self, text):
    # cleansing raw text
    model_input = self._clean_text(text)

    # vectorizing cleaned text
    model_input = self.tokenizer(text, return_tensors='pt', padding=True)
    return model_input

def inference(self, model_input):
    # get predictions from model as probabilities
    with torch.no_grad():
        model_output = self.model(*model_input.to(self.device))
        model_output = 1.0 / (1.0 + torch.exp(-model_output))
        model_output = model_output.numpy().astype(float)
    return model_output

def postprocess(self, model_output):
    # process predictions to predicted label and output format
    predicted_probabilities = model_output.max(axis=-1)
    predicted_ids = model_output.argmax(axis=-1)
    predicted_labels = [self._id2label[id_] for id_ in predicted_ids]
    return predicted_labels, predicted_probabilities

def handle(self, text):
    model_input = self.preprocess(text)
    model_output = self.inference(model_input)
    return self.postprocess(model_output)
```

딥러닝 모델도 마찬가지로 handler모델 코드를 작성해보자

머신러닝 모델과는 전처리와 vectorizing하는 방식이 조금 다르고 inference하는 부분도 머신러닝 같은 경우 sklearn 라이브러리에서 추상화를 잘 해놓아서 한줄의 코드라인으로도 inference할 수있게 되어있다면 딥러닝 모델 같은 경우 다른 라이브러리 등을 활용하면 비슷하게 짧게 구현할 수있겠지만 이번 실습에서는 토치 라이브러리를 사용해서 inference를 구현하였음.

구현한 것이 잘 동작하는지 터미널을 열어서 테스트를 진행함 테스트는 test_model_handler.py 모델을 돌려 테스트를 진행함 unittest 라이브러리를 사용하였으며 테스트 감성 문장들을 분류하는

task를 수행하게 된다

7. Flask 기반 감성분석 API 개발90

Unittest model handlers

개발한 model handler가 원했던 대로 동작하는지 테스트

```
$ python -m unittest -v test_model_handler.py
# test_ml_model_handler (test_model_handler.TestModelHandler) ... ok
# test_dl_model_handler (test_model_handler.TestModelHandler) ... ok
# -----
# Ran 2 tests in 11.815s
#
# OK
```

딥러닝 모델 핸들러 같은 경우 원격지에서 모델을 다운로드 하고 있는 상태이다. bert-base-multilingual-cased 모델을 기반으로 학습을 했기 때문에 모델 파일이 보이는 것처럼 712M여서 다운로드에 시간이 좀 걸렸다.

~ kdt-ai-aws [SSH: PROGRAMMERS]

> __pycache__

> data

└─ ratings_test.txt

└─ ratings_train.txt

> model

└─ app_fastapi.py

└─ app_flask.py

└─ client.py

└─ model.py

└─ README.md

└─ requirements.txt

└─ test_model_handler.py

└─ test.ipynb

└─ train_ml.py

└─ utils.py

62 def __init__(self):

63 super().__init__()

64 self.initialize()

65

66 def initialize(self,):

67 # Loading tokenizer and De-serializing model

68 from transformers import AutoTokenizer, AutoModelForSequenceClassification

69 from transformers.file_utils import is_torch_available

70 self.model_name_or_path = 'sacoh/bert-base-multilingual-cased-nsac'

71 self.tokenizer = AutoTokenizer.from_pretrained(self.model_name_or_path)

72 self.model = AutoModelForSequenceClassification.from_pretrained(self.model_name_or_path)

73

74 # if cuda is available, use GPU device

75 self.device = 'cuda:0' if is_torch_available else 'cpu'

76 self.model.to(self.device)

77

78 def preprocess(self, text):

79 # cleansing raw text

80 model_input = self._clean_text(text)

81

82 # vectorizing cleaned text

83 model_input = self.tokenizer(text, return_tensors='pt', padding=True)

84 return model_input

85

86 def inference(self, model_input):

87 # get predictions from model as probabilities

88 with torch.no_grad():

89 model_output = self.model(model_input.to(self.device))

90

문제

출력

디버그 콘솔

터미널

노트

JUPYTER

● (base) [ec2-user@ip-172-31-37-49 kdt-ai-aws]\$ conda activate kdt

● (kdt) [ec2-user@ip-172-31-37-49 kdt-ai-aws]\$ ls

app_fastapi.py client.py model __pycache__ requirements.txt test_model_handler.py utils.py

app_flask.py data model.py README.md test.ipynb train_ml.py

○ (kdt) [ec2-user@ip-172-31-37-49 kdt-ai-aws]\$ python test_model_handler.py

Downloading: 100% | 312/312 [00:00<00:00, 268kB/s]

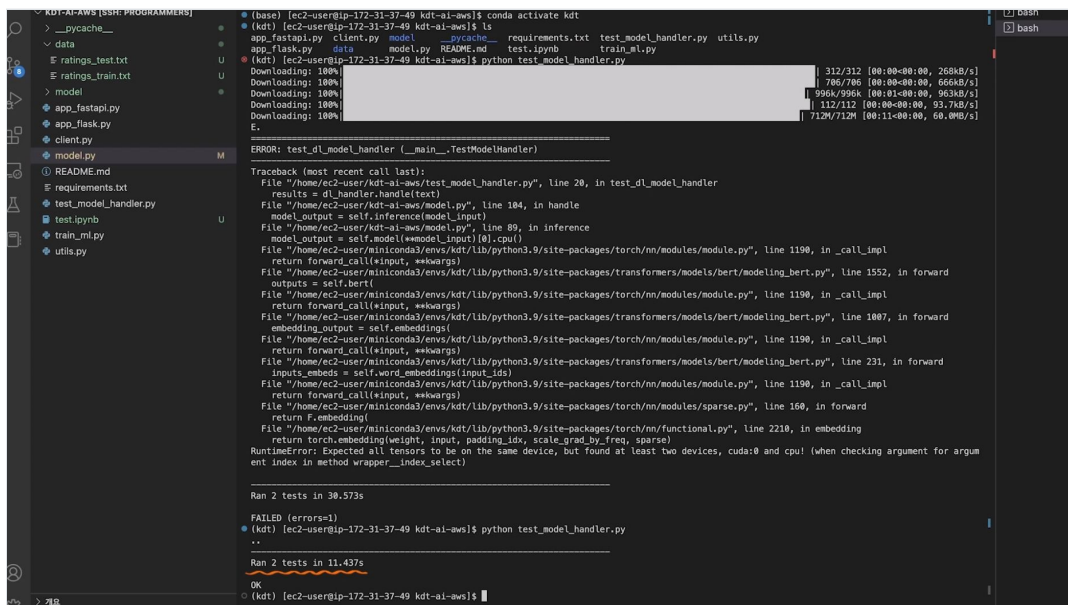
Downloading: 100% | 186/186 [00:00<00:00, 666kB/s]

Downloading: 100% | 998K/998K [00:01<00:00, 963kB/s]

Downloading: 100% | 112/112 [00:00<00:00, 93.7kB/s]

Downloading: 100% | 712M/712M [00:11<00:00, 60.0kB/s]

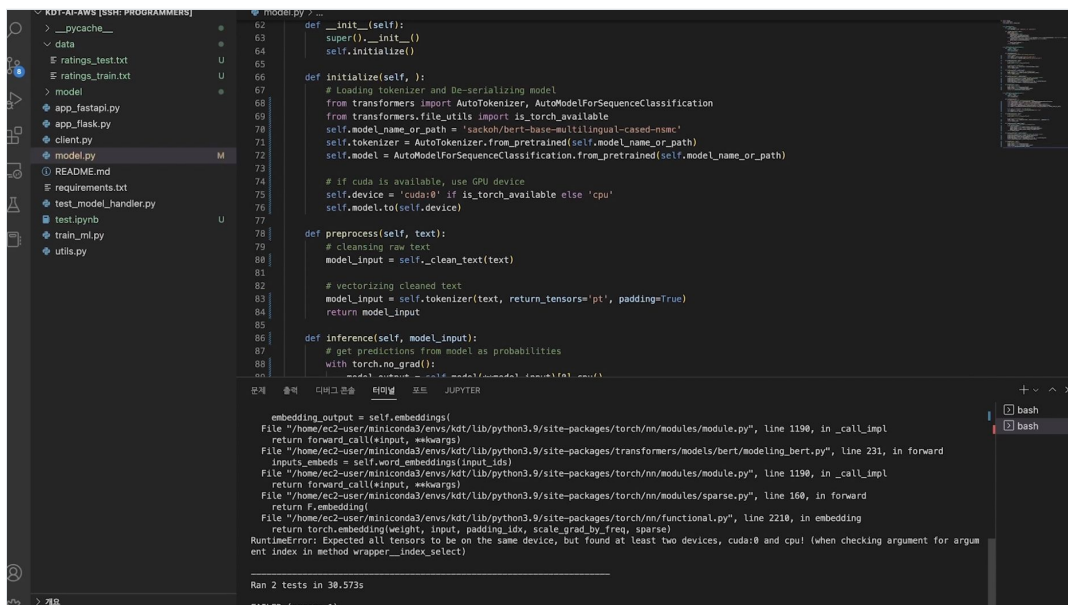
에러가 발생했는데 에러를 살펴보면 모든 텐서가 같은 device에 있어야 하는데 cuda:0와 cpu둘다에 올라갔다는 말이다. 그럼 우리가 핸들러를 제대로 구성하지 못했다는 것이고 모델은 현재 cuda에 올라갔는데 입력데이터가 cuda에 안올라 갔다는 말이다.



```
(base) [ec2-user@ip-172-31-37-49 kdt-ai-aws] conda activate kdt
(kdt) [ec2-user@ip-172-31-37-49 kdt-ai-aws] ls
app_fastapi.py client.py model _pycache_ requirements.txt test_model_handler.py utils.py
app_fastapi.py data model.py README.md test.ipynb train_ml.py
(kdt) [ec2-user@ip-172-31-37-49 kdt-ai-aws] python test_model_handler.py
Downloading: 100% | 312/312 [00:00<00:00, 268kB/s]
Downloading: 100% | 706/706 [00:00<00:00, 666kB/s]
Downloading: 100% | 998k/998k [00:01<00:00, 963kB/s]
Downloading: 100% | 112/112 [00:00<00:00, 93.7kB/s]
Downloading: 100% | 712M/712M [00:11<00:00, 60.0MB/s]
E-
ERROR: test_d1_model_handler (_main__TestModelHandler)
Traceback (most recent call last):
  File "/home/ec2-user/kdt-ai-aws/test_model_handler.py", line 20, in test_d1_model_handler
    results = d1_handler.handle(text)
  File "/home/ec2-user/kdt-ai-aws/model.py", line 184, in handle
    model_output = self.inference(model_input)
  File "/home/ec2-user/kdt-ai-aws/model.py", line 89, in inference
    model_output = self.model(model_input)[0].cpu()
  File "/home/ec2-user/miniconda3/envs/kdt/lib/python3.9/site-packages/torch/nn/modules/module.py", line 1190, in _call_impl
    return forward_call(*input, **kwargs)
  File "/home/ec2-user/miniconda3/envs/kdt/lib/python3.9/site-packages/transformers/models/bert/modeling_bert.py", line 1552, in forward
    outputs = self.bert(
  File "/home/ec2-user/miniconda3/envs/kdt/lib/python3.9/site-packages/torch/nn/modules/module.py", line 1190, in _call_impl
    return forward_call(*input, **kwargs)
  File "/home/ec2-user/miniconda3/envs/kdt/lib/python3.9/site-packages/transformers/models/bert/modeling_bert.py", line 1887, in forward
    embedding_output = self.embeddings(
  File "/home/ec2-user/miniconda3/envs/kdt/lib/python3.9/site-packages/torch/nn/modules/module.py", line 1190, in _call_impl
    return forward_call(*input, **kwargs)
  File "/home/ec2-user/miniconda3/envs/kdt/lib/python3.9/site-packages/transformers/models/bert/modeling_bert.py", line 231, in forward
    inputs_embeds = self.word_embeddings(input_ids)
  File "/home/ec2-user/miniconda3/envs/kdt/lib/python3.9/site-packages/torch/nn/modules/module.py", line 1190, in _call_impl
    return forward_call(*input, **kwargs)
  File "/home/ec2-user/miniconda3/envs/kdt/lib/python3.9/site-packages/torch/nn/modules/sparse.py", line 160, in forward
    return F.embedding(
  File "/home/ec2-user/miniconda3/envs/kdt/lib/python3.9/site-packages/torch/nn/functional.py", line 2210, in embedding
    return torch.embedding(weight, input, padding_idx, scale_grad_by_freq, sparse)
RuntimeError: Expected all tensors to be on the same device, but found at least two devices, cuda:0 and cpu! (when checking argument for argument index in method wrapper__index_select)

Ran 2 tests in 38.573s
FAILED (errors=1)
(kdt) [ec2-user@ip-172-31-37-49 kdt-ai-aws] python test_model_handler.py
Ran 2 tests in 11.437s
OK
(kdt) [ec2-user@ip-172-31-37-49 kdt-ai-aws] 
```

성공적으로 테스트가 완료된 모습이다! 테스트를 항상 할 수있도록 해야 내가 생각했던 코드가 실제로 동작하는지 확인을 하고 그 다음에 협업 저장소에 push를 할 수있을 것이다. 그래서 항상 test-case를 만들면서 개발하는 습관을 들이는데 중요하다!



```
class TestModelHandler:
    def __init__(self):
        super().__init__()
        self.initialize()

    def initialize(self, ):
        # Loading tokenizer and De-serializing model
        from transformers import AutoTokenizer, AutoModelForSequenceClassification
        from transformers.file_utils import is_torch_available
        self.model_name_or_path = 'sacoh/bert-base-multilingual-cased-nsmc'
        self.tokenizer = AutoTokenizer.from_pretrained(self.model_name_or_path)
        self.model = AutoModelForSequenceClassification.from_pretrained(self.model_name_or_path)

        # If cuda is available, use GPU device
        self.device = 'cuda:0' if is_torch_available else 'cpu'
        self.model.to(self.device)

    def preprocess(self, text):
        # Cleansing raw text
        model_input = self._clean_text(text)

        # Vectorizing cleaned text
        model_input = self.tokenizer(text, return_tensors='pt', padding=True)
        return model_input

    def inference(self, model_input):
        # get predictions from model as probabilities
        with torch.no_grad():
            model_output = self.model(**model_input).log_softmax(dim=-1)

        embedding_output = self.embeddings(
            File "/home/ec2-user/miniconda3/envs/kdt/lib/python3.9/site-packages/torch/nn/modules/module.py", line 1190, in _call_impl
            return forward_call(*input, **kwargs)
            File "/home/ec2-user/miniconda3/envs/kdt/lib/python3.9/site-packages/transformers/models/bert/modeling_bert.py", line 231, in forward
            inputs_embeds = self.word_embeddings(input_ids)
            File "/home/ec2-user/miniconda3/envs/kdt/lib/python3.9/site-packages/torch/nn/modules/module.py", line 1190, in _call_impl
            return forward_call(*input, **kwargs)
            File "/home/ec2-user/miniconda3/envs/kdt/lib/python3.9/site-packages/torch/nn/modules/sparse.py", line 160, in forward
            return F.embedding(
            File "/home/ec2-user/miniconda3/envs/kdt/lib/python3.9/site-packages/torch/nn/functional.py", line 2210, in embedding
            return torch.embedding(weight, input, padding_idx, scale_grad_by_freq, sparse)
RuntimeError: Expected all tensors to be on the same device, but found at least two devices, cuda:0 and cpu! (when checking argument for argument index in method wrapper__index_select)

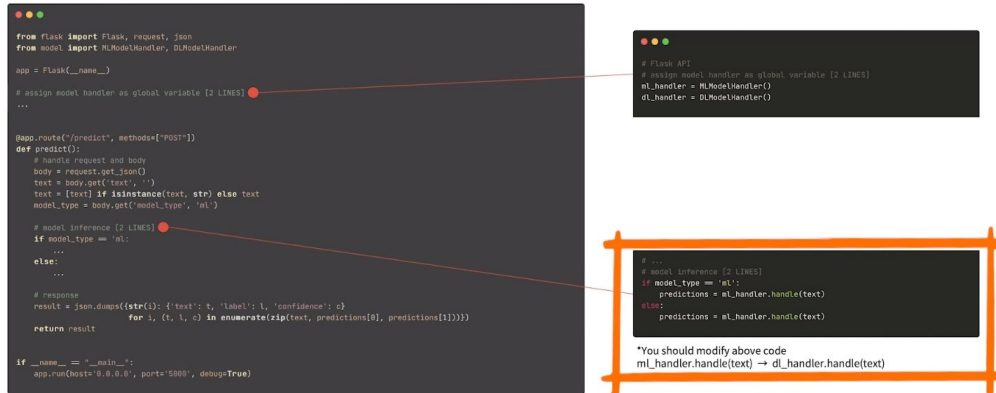
Ran 2 tests in 38.573s
FAILED (errors=1)
```

이제는 api개발과 배포를 진행할 것이다!

flask가 flask2로 업데이트 되면서 app.route()중간에 methjod=["POST"]로 지정을 해줘야 하는것도 있고 좀 더 직관적으로 API를 개발할 수있게 변경되었다. 그러나 여전에 micro web framework의 특성을 좀 더 집중해서 인지 그 외의 api로서의 사용할 기능들은 개발하려면 별도의 라이브러리를 사용해야 함

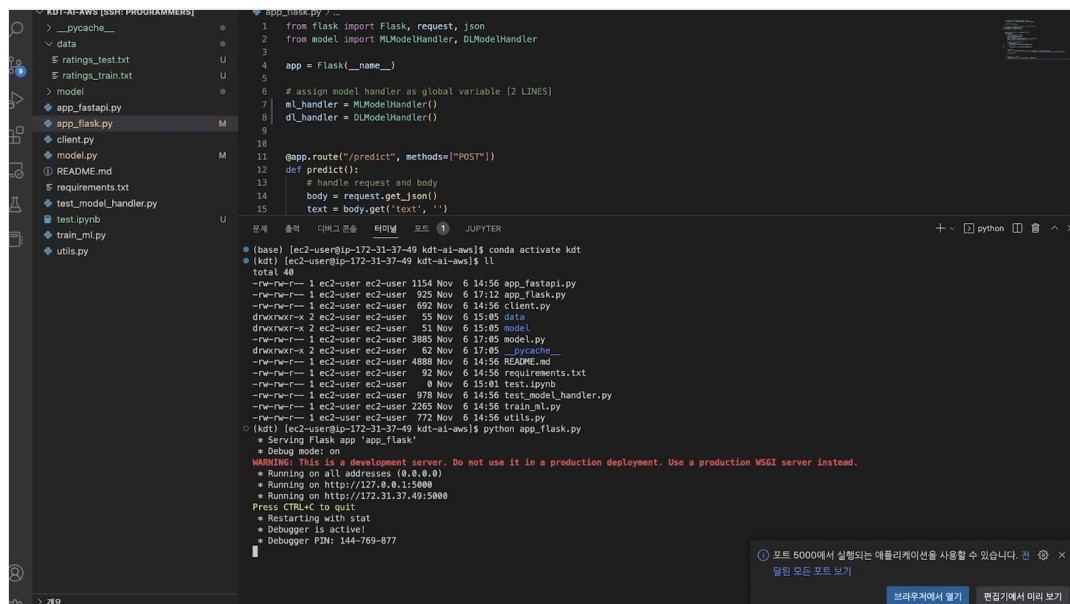
Flask API 개발 & 배포

Model을 전역변수로 불러오고 요청된 텍스트에 대해 예측 결과를 반환하는 코드 입력



flask같은 경우에는 request된 정보에 대해서 get_json()형태, 딕셔너리 형태로 값을 받을 수있다. body안에 key와 value형태로 request정보가 작성되어있고 우리가 처음에 api를 정의할 때, 텍스트와 모델 타입, 이 2가지가 반드시 들어갈 수있도록 api를 정의 했었다. body에서 text와 모델 타입을 불러온다. 그래서 모델 타입이 ml일 경우 ml_handler로 dl일 경우 dl_handler로 사용하도록 한다

app_flask.py를 실행시키면 flask앱을 서버로서 실행이 되는데 그 때 주소를 0.0.0.0으로해서 모든 ip에서 로컬호스트에서 접근할 수있도록 했고 포트는 5000으로 설정했다.



빨간색 warning을 보면 WSGI가 나오는데 이것은 웹 프레임워크에서 사용하는 프로토콜입니다. 잘 실행이 되는 모습을 확인할 수있고 이제 테스트를 해보자. 테스트는 2가지 방식으로 진행할 수있다.

Test API on remote

원격에서 서버로 API에 요청하여 테스트 수행

host: EC2 인스턴스 생성 시에 받은 퍼블릭 IP 주소 또는 직접 할당한 탄력적 IP 주소 (ex. 54.83.1.129)

port : EC2 인스턴스 생성 시에 설정했던 port 번호 (ex. 5000)

```
$ curl -d '{"text": ["영화 오랜만에 봤는데 괜찮은 영화였어", "정말 지루했어"], "model_type": "ml"}' \
-H 'Content-Type: application/json' \
-X POST \
http://{HOSTNAME}:{PORT}/predict
```

```
import request

url = f'http://{HOSTNAME}:{PORT}/predict'
data = {"text": ["영화 오렌지엔 밤는데 편식은 영화였어", "정말 지루했어"], "model_type": "ml"}
response = requests.post(url, json=data)
print(response.content)
```

*You should modify above code
 uson=data → json=data

먼저 터미널 환경에서 로컬로 호출하는 방식으로 해보고 파이썬 스크립트 환경에서 테스트를 해보려고 한다.

매직파이썬 키워드를 사용하면 쥬피터 노트북에서도 bash의 명령어를 사용할 수 있다!

The screenshot shows a JupyterLab environment. On the left, a file explorer lists files including `ratings_train.txt`, `test_model_handler.py`, and `utils.py`. The main workspace contains a terminal window with the following content:

```
curl -d '{"text": "[\"영화 오랑엔가 뽀논데 겐뽀은 영화영어\", \"장영 지루영어\"]\", \"model_type\": \"ml\"}' \
  -H \"Content-Type: application/json\" \
  -X POST -v \
  http://localhost:5000/predict
```

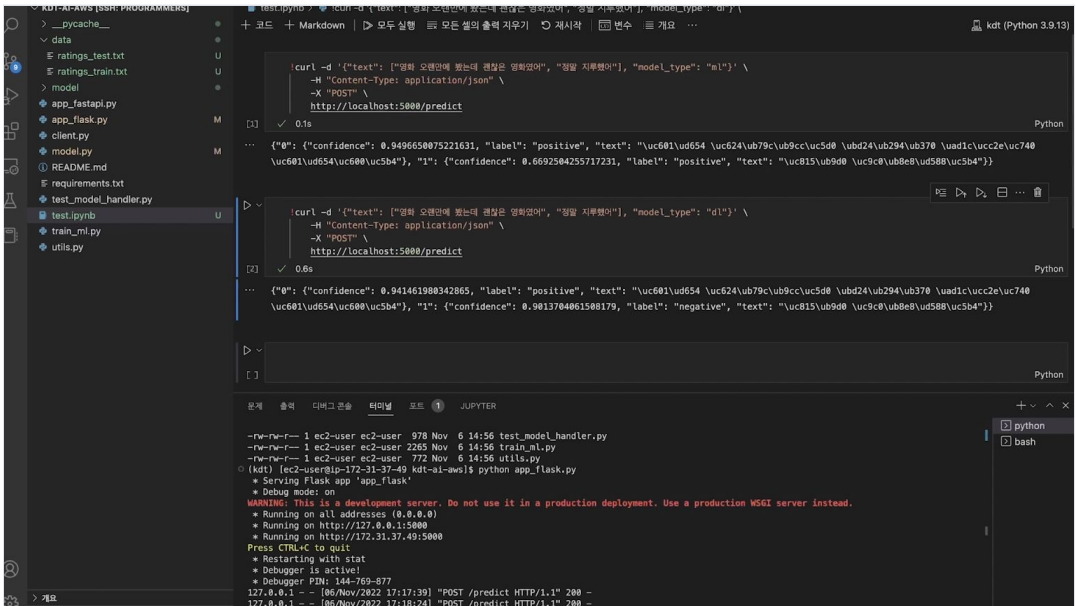
The output of the curl command is displayed below:

```
{\"confidence\": 0.9496650875221631, \"label\": \"positive\", \"text\": \"\uc601\uc654 \uc624\uc79c\uc69c\uc5d8 \uc6d24uc294uc378 \uc61c\uc62e\uc740 \uc601\uc654\uc608\uc5b4\"\", \"i\": {\"confidence\": 0.6692584255717231, \"label\": \"positive\", \"text\": \"\uc6b5\uc69d \uc6c9c\uc6b8\uc588\uc5b4\"}}
```

Below the main terminal, a second terminal window shows system logs and a warning:

```
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://172.17.0.1:5000
* Running on http://172.31.37.49:5000
Press CTRL-C to quit
* Restarting with stat
* Debugger is active!
* Debugger PID: 144-769-877
```

맨 밑에 200코드는 성공적으로 응답을 했다는 코드이다 결과를 보면 첫번째도 긍정 두번째 문장도 긍정(?)으로 답한 모습이다. 이번에는 딥러닝 모델로 한번 해보자 딥러닝 모델은 첫번째 긍정, 두번째는 부정으로 답한다. 근대 응답하는 속도가 머신러닝 모델이 속도가 조금 더 빠르다.



7. Flask 기반 감성분석 API 개발

92

Test API on remote

원격에서 서버로 API에 요청하여 테스트 수행

host: EC2 인스턴스 생성 시에 받은 퍼블릭 IP 주소 또는 직접 할당한 탄력적 IP 주소 (ex. 54.83.1.129)

port: EC2 인스턴스 생성 시에 설정했던 port 번호 (ex. 5000)



과제

Train API 추가

ML model 학습 요청 API 개발

Serialization 실습에서 실행했던 `train_ml.py` 참조하여

<http://host:port/train> 으로 모델 학습을 요청하는 API 추가

