



챗봇 프로젝트 10

라이브러리 설치

텐서플로우 2.2 – 딥러닝 모델 실습

```
pip install tensorflow==2.2
```

사이킷런 – 머신러닝 도구 제공

```
pip install sklearn ( 혹은, pip install scikit-learn)
```

Seqeval – 모델의 평가를 위한 라이브러리

```
pip install seqeval
```

판다스 – 데이터 분석 및 처리

```
pip install pandas xlrd
```

Matplotlib – 데이터 시각화 도구 제공

```
pip install matplotlib==3.6
```

MySQL

MySQL은 가장 많이 사용되는 오픈소스 관계형 데이터베이스 관리 시스템(RDBMS)이다.

챗봇 시스템의 학습 데이터 관리를 위해 MySQL을 사용.

파이썬에서 MySQL을 어떻게 연동하고 사용하는지 알아보고,

이를 활용해 학습툴을 만들어본다.

(chatbot2)

`pip install PyMySQL`

`pip install openpyxl`

명령어	설명
select	데이터 테이블에서 데이터를 조회합니다.
insert	데이터 테이블에 데이터를 삽입합니다.
update	데이터 테이블의 데이터를 변경합니다.
delete	데이터 테이블의 데이터를 삭제합니다.

MySQL 설치

<https://dev.mysql.com/downloads/windows/installer/8.0.html>

MySQL 다운로드 후 실행


Developer Default 선택 후, 설치 진행

MySQL Installer 8.0.33

Select Operating System:
Microsoft Windows

Looking for previous GA versions?

Windows (x86, 32-bit), MSI Installer	8.0.33	2.4M	Download
(mysql-installer-web-community-8.0.33.0.msi) MD5: 2a330cf24915964cca87e04dbb34e5d3 Signature			
Windows (x86, 32-bit), MSI Installer	8.0.33	428.3M	Download
(mysql-installer-community-8.0.33.0.msi) MD5: 3d4c833d05a6e0a330bca11d4d1cc Signature			

 We suggest that you use the [MD5 checksums](#) and [GnuPG signatures](#) to verify the integrity of the packages you download.

설치 과정 중, 비밀번호 설정 '1234'

설치 완료 후, MySQL 8.0 Command line Client 실행

`show databases;` 입력 시 현재 데이터베이스 목록 출력

`create database homestead;` homestead라는 이름의 데이터베이스 생성

```
mysql> create schema homestead
-> ;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| homestead |
| information_schema |
| mysql |
| performance_schema |
| sakila |
| sys |
| world |
+-----+
```

db-conn.py

```
import pymysql
db = None
try:
    db = pymysql.connect(
        host='127.0.0.1',
        user='root',
        passwd='1234',
        db='homestead',
        charset='utf8'
    )
    print("DB 연결 성공")

except Exception as e:
    print(e)
finally:
    if db is not None:
        db.close()
        print("DB 연결 닫기 성공")
```

```
(chatbot2) C:\Users\yubeen\python-chatbot\DB>python db_conn.py  
DB 연결 성공  
DB 연결 닫기 성공
```

DB 정상 연결 시,

DB 연결 성공

DB 연결 닫기 성공

메시지 출력

make-table.py

```
import pymysql

db = None
try:
    db = pymysql.connect(
        host='127.0.0.1',
        user='root',
        passwd='1234',
        db='homestead',
        charset='utf8'
    )
    sql = '''
CREATE TABLE tb_student (
    id int primary key auto_increment not null,
    name varchar(32),
    age int,
    address varchar(32)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
'''
```



```
with db.cursor() as cursor:
    cursor.execute(sql)

except Exception as e:
    print(e)

finally:
    if db is not None:
        db.close()
```

실행 후, mysql cli client에서

`use homestead;` 만든 DB 선택

`show tables;` 선택한 DB 안에 있는 테이블 조회

tb_student 테이블이 생성되어있으면 성공.

```
mysql> show tables;
+-----+
| Tables_in_homestead |
+-----+
| tb_student          |
+-----+
1 row in set (0.00 sec)
```

insert-data.py

```
import pymysql

db = None
try:
    db = pymysql.connect(
        host='127.0.0.1',
        user='root',
        passwd='1234',
        db='homestead',
        charset='utf8'
    )
    sql = '''
    INSERT tb_student(name, age, address) values('Kei', 35,
    'Korea')
    '''
    with db.cursor() as cursor:
        cursor.execute(sql)
    db.commit()
```

```
except Exception as e:  
    print(e)  
  
finally:  
    if db is not None:  
        db.close()
```

실행 후, mysql cli client에서

```
select * from tb_student;
```

선택한 테이블에서 데이터 전체를 조회

코드 내 sql 문에 입력한 데이터가 저장됐으면 성공

```
mysql> select * from tb_student  
-> ;  
+----+-----+-----+-----+  
| id | name | age  | address |  
+----+-----+-----+-----+  
|  1 | Kei  |   35 | Korea   |  
+----+-----+-----+-----+  
1 row in set (0.00 sec)
```

update-data.py

```
import pymysql

db = None
try:
    db = pymysql.connect(
        host='127.0.0.1',
        user='root',
        passwd='1234',
        db='homestead',
        charset='utf8'
    )
    id = 1  # E//O/E/ id (PK)
    sql = '''
        UPDATE tb_student set name="케이", age=36 where id=%d
        ''' % id
    with db.cursor() as cursor:
        cursor.execute(sql)
    db.commit()
```

```
except Exception as e:
    print(e)

finally:
    if db is not None:
        db.close()
```

실행 후, mysql cli client에서

```
select * from tb_student;
```

선택한 테이블에서 데이터 전체를 조회

기존 데이터에서 이름 나이의 데이터가
수정 되었으면 성공.

```
mysql> select * from tb_student;
+----+-----+-----+-----+
| id | name | age  | address |
+----+-----+-----+-----+
|  1 | 케이 |   36 | Korea   |
+----+-----+-----+-----+
1 row in set (0.00 sec)
```

delete-data.py

```
import pymysql

db = None
try:
    db = pymysql.connect(
        host='127.0.0.1',
        user='root',
        passwd='1234',
        db='homestead',
        charset='utf8'
    )
    id = 1  # E//O/E/ id (PK)
    sql = '''
        DELETE from tb_student where id=%d
        ''' % id
    with db.cursor() as cursor:
        cursor.execute(sql)
    db.commit()
```

```
except Exception as e:  
    print(e)  
  
finally:  
    if db is not None:  
        db.close()
```

실행 후, mysql cli client에서

```
select * from tb_student;
```

선택한 테이블에서 데이터 전체를 조회

테이블 데이터가 삭제 되었으면 성공.

```
mysql> select * from tb_student;  
Empty set (0.00 sec)
```

select-data.py

```
import pymysql
import pandas as pd

db = None
try:
    db = pymysql.connect(
        host='127.0.0.1',
        user='root',
        passwd='1234',
        db='homestead',
        charset='utf8'
    )
    students = [
        {'name': 'Kei', 'age': 36, 'address': 'PUSAN'},
        {'name': 'Tony', 'age': 34, 'address': 'PUSAN'},
        {'name': 'Jaeyoo', 'age': 39, 'address': 'GWANGJU'},
        {'name': 'Grace', 'age': 28, 'address': 'SEOUL'},
        {'name': 'Jenny', 'age': 27, 'address': 'SEOUL'},
    ]
```



```
for s in students:
    with db.cursor() as cursor:
        sql = '''
            insert tb_student(name, age, address)
values("%s",%d,"%s")
            ''' % (s['name'], s['age'], s['address'])
        cursor.execute(sql)
    db.commit()

cond_age = 30
with db.cursor(pymysql.cursors.DictCursor) as cursor:
    sql = '''
    select * from tb_student where age > %d
    ''' % cond_age
    cursor.execute(sql)
    results = cursor.fetchall()
print(results)
```

```
cond_name = 'Grace'
with db.cursor(pymysql.cursors.DictCursor) as cursor:
    sql = '''
        select * from tb_student where name="%s"
        ''' % cond_name
    cursor.execute(sql)
    result = cursor.fetchone()
    print(result['name'], result['age'])

df = pd.DataFrame(results)
print(df)
```

```
except Exception as e:
    print(e)
```

```
finally:
    if db is not None:
        db.close()
```

```
(chatbot2) C:\Users\yubeen\python-chatbot\DB>python select-data.py
[{'id': 2, 'name': 'Kei', 'age': 36, 'address': 'PUSAN'}, {'id': 3, 'name': 'Tony', 'age': 34, 'address': 'PUSAN'}, {'id': 4, 'name': 'Jaeyoo', 'age': 39, 'address': 'GWANGJU'}]
Grace 28
  id  name  age  address
0   2   Kei   36   PUSAN
1   3  Tony   34   PUSAN
2   4 Jaeyoo  39  GWANGJU
```

코드 실행 시, 테이블에 다수의 데이터 추가 후,

조건에 맞는 데이터를 출력

mysql cli client에서

`select * from tb_student;`

선택한 테이블에서 데이터 전체를 조회

추가한 다수의 데이터 컬럼 확인 가능.

```
mysql> select * from tb_student;
+----+-----+-----+-----+
| id | name  | age  | address |
+----+-----+-----+-----+
| 2  | Kei   | 36   | PUSAN   |
| 3  | Tony  | 34   | PUSAN   |
| 4  | Jaeyoo | 39   | GWANGJU |
| 5  | Grace | 28   | SEOUL   |
| 6  | Jenny | 27   | SEOUL   |
+----+-----+-----+-----+
5 rows in set (0.00 sec)
```

실습. User 데이터베이스를 생성, user_info 테이블을 생성하여 아래 표와 같이 데이터를 입력해보자
(데이터베이스, 테이블, 컬럼 등의 이름은 임의대로 지어도 상관없음)

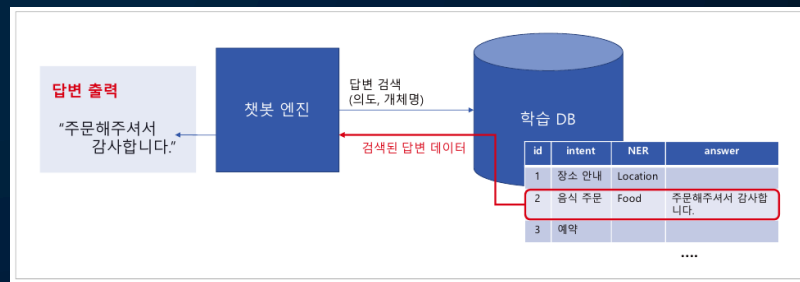
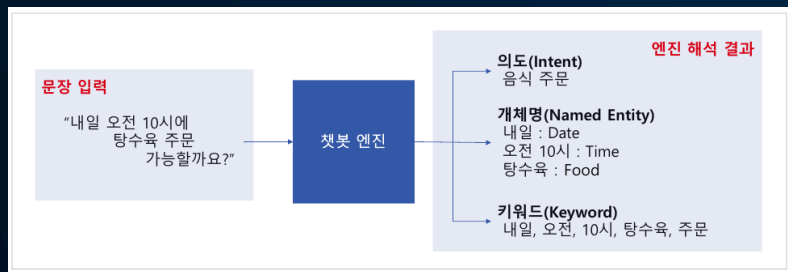
이름	성별	나이	전화번호	국적
김철수	남자	15	010-1234-5678	한국
홍길동	남자	35	010-2222-3333	한국
Jason	남자	21	010-1212-2323	미국
이영희	여자	22	010-9876-1234	한국
Emma	여자	27	010-4567-7890	스페인

챗봇 학습툴 만들기

MySQL로 챗봇의 학습 데이터를 관리하는 툴을 제작

학습데이터를 DB에 저장했을 때 실시간으로 챗봇 시스템에 적용될 수 있도록 구현

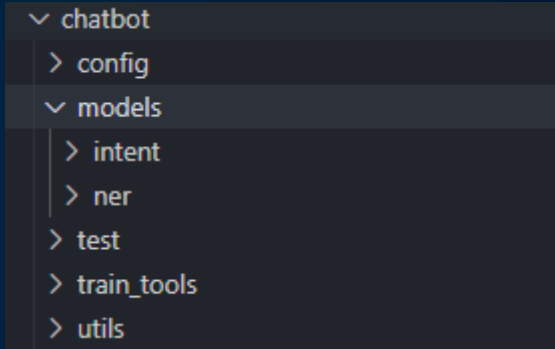
이를 위해 챗봇이 이해할 수 있는 질문 정보와 해당 답변 데이터를 관리하기 위한 툴이 필요.



폴더 구조

챗봇 구현 과정 시 파일 관리를 위해 아래와 같이 폴더를 생성

```
chatbot
├── config
├── models
│   ├── intent
│   └── ner
├── test
├── train_tools
└── util
```



```
▼ chatbot
  > config
  ▼ models
    > intent
    > ner
  > test
  > train_tools
  > utils
```

A screenshot of a file explorer window showing the directory structure for a chatbot project. The 'chatbot' directory is expanded, revealing subdirectories: 'config', 'models' (which is further expanded to show 'intent' and 'ner'), 'test', 'train_tools', and 'utils'.

DatabaseConfig.py

```
DB_HOST = "127.0.0.1"
DB_USER = "root"
DB_PASSWORD = "1234"
DB_NAME = "homestead"

def DatabaseConfig():
    global DB_HOST, DB_USER, DB_PASSWORD, DB_NAME
```

코드 작성 후 /config 폴더 안에 저장

DB 접속 정보는 본인 환경에 맞춰서 작성

DB 설계, DB 생성

아래 표와 같은 형식의 테이블을 생성

테이블 명 : chatbot_train_data

컬럼	속성	설명
id	int primary key not null	학습 데이터 id
intent	varchar(45)	의도명, 의도가 없는 경우 null
ner	varchar(45)	개체명, 개체명이 없는 경우 null
query	text null	질문 텍스트
answer	text not null	답변 텍스트
answer_image	varchar(2048)	답변에 들어갈 이미지 URL, 이미지 URL을 사용하지 않을 경우 null

create_train_data_table.py

/train_tools/qna 폴더 생성 후, qna 폴더 안에 코드 파일 작성

```
import pymysql
import sys
sys.path.append('../..')
from config.DatabaseConfig import *

db = None
try:
    db = pymysql.connect(
        host='127.0.0.1',
        user='root',
        passwd='1234',
        db='homestead',
        charset='utf8'
    )
```

```
sql = '''
    CREATE TABLE IF NOT EXISTS `chatbot_train_data` (
        `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
        `intent` VARCHAR(45) NULL,
        `ner` VARCHAR(1024) NULL,
        `query` TEXT NULL,
        `answer` TEXT NOT NULL,
        `answer_image` VARCHAR(2048) NULL,
        PRIMARY KEY(`id`))
ENGINE = InnoDB DEFAULT CHARSET=utf8
'''

with db.cursor() as cursor:
    cursor.execute(sql)
db.commit()

except Exception as e:
    print(e)

finally:
    if db is not None:
        db.close()
```

코드 실행 시, 테이블 구조 생성

`show tables;`

DB안의 테이블 목록 출력

`desc chatbot_train_data;`

chatbot_train_data 테이블의 구조 출력

실행 성공 시,

테이블이 추가됨과 동시에

데이터를 담을 컬럼 구조가 생성된다.

```
mysql> show tables;
+-----+
| Tables_in_homestead |
+-----+
| chatbot_train_data   |
| tb_student           |
+-----+
2 rows in set (0.00 sec)
```

```
mysql> desc chatbot_train_data;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra           |
+-----+-----+-----+-----+-----+-----+
| id         | int unsigned  | NO   | PRI | NULL    | auto_increment |
| intent     | varchar(45)   | YES  |     | NULL    |                 |
| ner        | varchar(1024) | YES  |     | NULL    |                 |
| query      | text          | YES  |     | NULL    |                 |
| answer     | text          | NO   |     | NULL    |                 |
| answer_image | varchar(2048) | YES  |     | NULL    |                 |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

데이터 엑셀 파일 및 DB연동

	A	B	C	D	E
1	의도(Intent)	개체명(NER)	질문(Query)	답변(Answer)	답변 이미지
				네 안녕하세요 :D	
인사			안녕하세요	네 안녕하세요 :D 반갑습니다. 저는 챗봇입니다.	https://i.imgur.com/UluUFMp.jpg
인사			반가워요	네 안녕하세요 :D 반갑습니다. 저는 챗봇입니다.	https://i.imgur.com/UluUFMp.jpg
주문	B_FOOD	{B_FOOD}	{B_FOOD} 주문할게요	{B_FOOD} 주문 처리 완료되었습니다. 주문해주셔서 감사합니다.	
주문	B_FOOD	{B_FOOD}	{B_FOOD} 주문할게요	{B_FOOD} 주문 처리 감사!!	
예약	B_DT,B_TI	{B_DT}	{B_DT} 예약	{B_DT}에 예약 접수 되었습니다.	
욕설				욕하면 나빠요 ㅠ	

data_load.py

qna 폴더 안에 코드 파일 작성

```
import pymysql
import openpyxl
import sys
sys.path.append('../..')
from config.DatabaseConfig import *

def all_clear_train_data(db):
    sql = '''
        delete from chatbot_train_data
    '''
    with db.cursor() as cursor:
        cursor.execute(sql)

    sql = '''
        ALTER TABLE chatbot_train_data AUTO_INCREMENT=1
    '''
    with db.cursor() as cursor:
        cursor.execute(sql)
```

```
def insert_data(db, xls_row):
    intent, ner, query, answer, answer_img_url = xls_row

    sql = '''
        INSERT chatbot_train_data(intent, ner, query, answer,
answer_image)
        values(
            '%s', '%s', '%s', '%s', '%s'
        )
    ''' % (intent.value, ner.value, query.value, answer.value,
answer_img_url.value)

    sql = sql.replace("'None'", "null")

    with db.cursor() as cursor:
        cursor.execute(sql)
        print('{} 저장'.format(query.value))
        db.commit()
```

```
train_file = './train_data.xlsx'
db = None
try:
    db = pymysql.connect(
        host='127.0.0.1',
        user='root',
        passwd='1234',
        db='homestead',
        charset='utf8'
    )

    all_clear_train_data(db)

    wb = openpyxl.load_workbook(train_file)
    sheet = wb['Sheet1']
    for row in sheet.iter_rows(min_row=2):
        insert_data(db, row)

    wb.close()

except Exception as e:
    print(e)

finally:
    if db is not None:
        db.close()
```

코드 실행 시, 엑셀 파일의 데이터가

MySQL DB에 저장된 것을 확인할 수 있음.

```
mysql> select * from chatbot_train_data;
```

id	intent	ner	query	answer	answer_image
1	인사	NULL	안녕하세요	네 안녕하세요 :D 반갑습니다. 저는 챗봇입니다.	https://i.imgur.com/UluUFMp.jpg
2	인사	NULL	반가워요	네 안녕하세요 :D 반갑습니다. 저는 챗봇입니다.	https://i.imgur.com/UluUFMp.jpg
3	주문	B_FOOD	{B_FOOD} 주문할게요	{B_FOOD} 주문 처리 완료되었습니다.	
4	주문	B_FOOD	{B_FOOD} 주문할게요	{B_FOOD} 주문 처리 감사!!	NULL
5	예약	B_DT,B_TI	{B_DT} 예약	{B_DT}에 예약 접수 되었습니다.	NULL
6	욕설	NULL	NULL	욕하면 나빠요 ㅠ	NULL

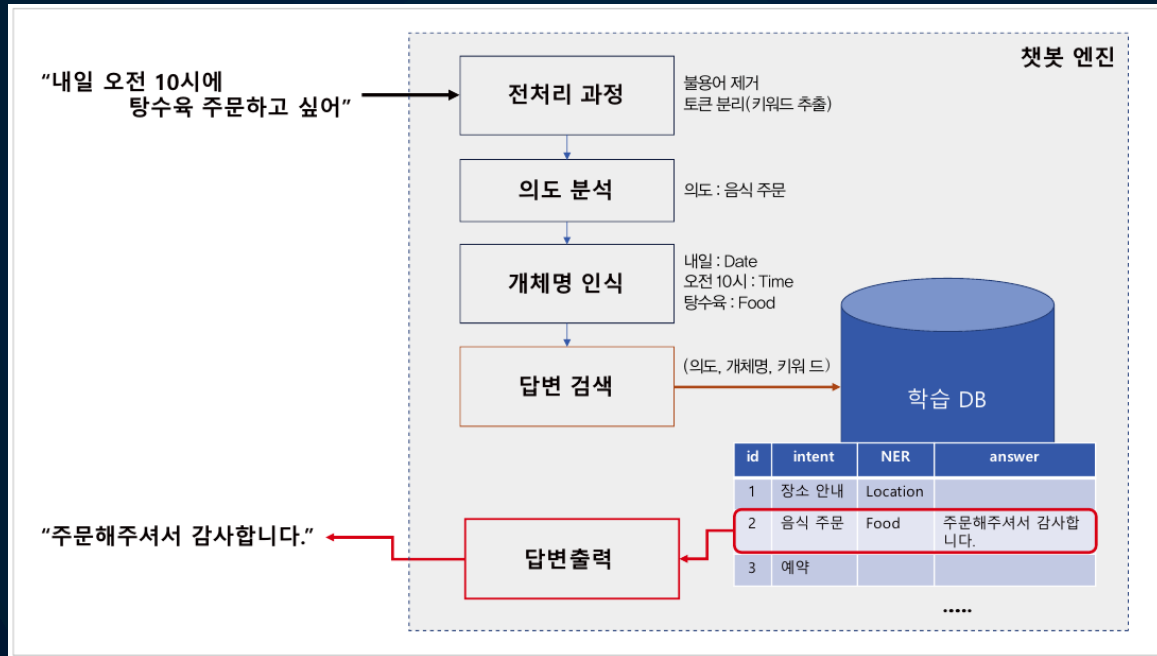
```
6 rows in set (0.00 sec)
```


챗봇 엔진

챗봇 엔진 핵심 기능

핵심 기능	설명
질문 의도 분류	화자의 질문 의도를 파악. 해당 질문을 의도 분류 모델을 이용해 의도를 예측한다.
개체명 인식	화자의 질문에서 단어 토큰별 개체명을 인식. 개체명을 예측한다.
핵심 키워드 추출	화자의 질문 의미에서 핵심이 될 만한 단어 토큰을 추출. 현재소 분석기를 이용해 핵심 키워드가 되는 명사, 동사를 추출한다.
답변 검색	해당 질문의 의도, 개체명, 핵심 키워드 등을 기반으로 답변을 학습 DB에서 검색.
서버	다양한 종류(카카오톡, 네이버톡톡) 등 연동 시, 챗봇 클라이언트에서 요청하는 질문을 처리하기 위해 소켓 서버 프로그램 역할을 한다.

엔진 처리 과정



Preprocess.py

전처리 과정을 위한 코드
util 폴더 안에 코드 작성

```
from konlpy.tag import Komoran

class Preprocess:
    def __init__(self, word2index_dic='', userdic=None):
        self.komoran = Komoran(userdic=userdic)

        self.exclusion_tags = [
            'JKS' , 'JKC', 'JKG', 'JKO', 'JKB', 'JKV', 'JKQ',
            'JX', 'JC',
            'SF', 'SP', 'SS', 'SE', 'SO',
            'EP', 'EF', 'EC', 'ETN', 'ETM',
            'XSN', 'XSV', 'XSA'
        ]
```

```
def pos(self, sentence):  
    return self.komoran.pos(sentence)  
  
def get_keywords(self, pos, without_tag=False):  
    f = lambda x: x in self.exclusion_tags  
    word_list=[]  
    for p in pos:  
        if f(p[1]) is False:  
            word_list.append(p if without_tag is  
False else p[0])  
    return word_list
```

공유한 user_dic.tsv 파일 util 경로 안에 저장.

전처리 기능을 구현한 클래스 생성.

클래스 테스트 코드는 추가 작성 필요.

Preprocess_test.py

test 폴더 안에 코드 작성

```
import sys
sys.path.append('../')
from util.Preprocess import Preprocess

sent = "내일 1시에 짜장면 먹으러가자"

p = Preprocess(userdic='user_dic.tsv')

pos = p.pos(sent)

ret = p.get_keywords(pos, without_tag=False)
print(ret)

ret = p.get_keywords(pos, without_tag=True)
print(ret)
```

```
(chatbot2) C:\Users\yubeen\python-chatbot\chatbot\test>python Preprocess_test.py  
[('내일', 'NNG'), ('1시', 'NNG'), ('짜장면', 'NNG'), ('먹', 'V'), ('가', 'VX')]  
['내일', '1시', '짜장면', '먹', '가']
```

Preprocess 클래스 안에 구현한

형태소 분석 함수 pos와

키워드를 출력해주는 get_keywords 함수의 작동을 확인

단어 사전 구축

앞의 실습에서 사용했던 말뭉치 데이터를 합쳐서 단어 사전 구축.

말뭉치 데이터 (corpus.txt) 파일을 /train_tools/dict/ 경로에 저장.

create_dict.py

train_tools/dict 폴더 안에 코드 작성

```
import sys
sys.path.append('../..')
from util.Preprocess import Preprocess
from tensorflow.keras import preprocessing
import pickle

def read_corpus_data(filename):
    with open(filename, 'r', encoding='UTF-8') as f:
        data = [line.split('\t') for line in
f.read().splitlines()]
    return data

corpus_data = read_corpus_data('./corpus.txt')

p = Preprocess(word2index_dic='chatbot_dict.bin',
                userdic = '../..util/user_dic.tsv')
```



```
dict = []
for c in corpus_data:
    pos = p.pos(c[1])
    for k in pos:
        dict.append(k[0])

tokenizer = preprocessing.text.Tokenizer(oov_token='OOV')
tokenizer.fit_on_texts(dict)
word_index = tokenizer.word_index

f = open("chatbot_dict.bin", "wb")
try:
    pickle.dump(word_index, f)
except Exception as e:
    print(e)
finally:
    f.close()
```

실행 시, train_tools/dict 경로에 chatbot_dict.bin 이름의 단어 사전이 생성

chatbot_dict_test.py

test 폴더 안에 코드 작성

```
import pickle
import sys
sys.path.append('../')
from util.Preprocess import Preprocess

f = open("../train_tools/dict/chatbot_dict.bin", "rb")
word_index = pickle.load(f)
f.close()

sent = "내일 오후 1시에 짜장면 먹으러 가자 ㅋㅋ"

p = Preprocess(userdic='../util/user_dic.tsv')

pos = p.pos(sent)

keywords = p.get_keywords(pos, without_tag=True)
```

```
keywords = p.get_keywords(pos, without_tag=True)
for word in keywords:
    try:
        print(word, word_index[word])
    except KeyError:
        print(word, word_index['OOV'])
```

방금 생성한 단어 인덱스 사전 파일을 불러와서 테스트.

각 단어에 대한 index 값을 출력한다.

입력한 문장을 단어 인덱스 사전을 이용하여 단어 시퀀스 벡터로

변환하는 기능을 기존 util/Preprocess.py 코드에 추가

```
(chatbot2) C:\Users\yut
내일 12
오후 127
1시 94
짜장면 321
먹 178
가자 3795
ㅋㅋ 10844
```

Preprocess.py

```
from konlpy.tag import Komoran
import pickle

class Preprocess:
    def __init__(self, word2index_dic='', userdic=None):
        self.komoran = Komoran(userdic=userdic)
        # 단어 인덱스 사전 불러오기
        if(word2index_dic != ''):
            f = open(word2index_dic, "rb")
            self.word_index = pickle.load(f)
            f.close()
        else:
            self.word_index = None

        self.exclusion_tags = [
            ...생략
        ]
```

```
def pos(self, sentence):  
    ...  
def get_keywords(self,pos,without_tag=False):  
    ...  
  
def get_wordidx_sequence(self, keywords):  
    if self.word_index is None:  
        return []  
  
    w2i = []  
    for word in keywords:  
        try:  
            w2i.append(self.word_index[word])  
        except KeyError:  
            w2i.append(self.word_index['OOV'])  
    return w2i
```

의도 분류 모델

화자가 문장을 입력했을 때, 전처리 과정을 거친 후, 해당 문장의 의도를 분류한다.
문장 의도 분류는 CNN 모델을 사용.
데이터가 한정적이기 때문에 5가지 의도만 분류

의도명	분류 클래스	설명
인사	0	텍스트가 인사말인 경우
욕설	1	텍스트가 욕설인 경우
주문	2	텍스트가 주문 관련 내용인 경우
예약	3	텍스트가 예약 관련 내용인 경우
기타	4	어떤 의도에도 포함되지 않은 경우

GlobalParams.py

config 폴더 안에 코드 작성

```
MAX_SEQ_LEN = 15

def GlobalParams():
    global MAX_SEQ_LEN
```

모델 학습 모델을 만들기 전, 챗봇 엔진 소스코드 전역에서 사용할 파라미터의 정보를
관리할

파일 GlobalParams.py를 작성

의도 분류 모델 학습

모델의 설계 및 학습을 진행

/models/intent 경로에 total_train_data.csv 파일을 저장.

해당 파일을 데이터셋으로 사용한다.

```
1 query,intent
2 헬로우,0
3 헬로,0
4 안부 인사드립니다.,0
5 먼저 인사하려고 했는데 짝남이 먼저 인사해줬어. 더 떨렸겠어요.,0
6 먼저 인사할까 했는데 짝녀가 먼저 인사해줬어. 기분 좋았겠네요.,0
7 각자 집에 인사드리러 가 제가 더 떨리네요.,0
8 이별의 마무리 가벼운 안부 인사일 거예요.,0
9 용기내서 새해인사 했네 이제 연락하지 마세요.,0
10 오늘 마지막 인사를 하러가네 미련없길 바랄게요.,0
11 오늘 마지막 인사하러 갑니다. 미련없이 정리했긴 바랍니다.,0
12 내 마음에 마지막 인사 마음이랑 잘 인사해요.,0
13 먼저 잘게 잘자 안녕히 주무세요.,0
```


train_model.py

models/intent 폴더 안에 코드 작성

```
import sys
sys.path.append('../..')
import pandas as pd
import tensorflow as tf
from tensorflow.keras import preprocessing
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Dense,
Dropout, Conv1D, GlobalMaxPool1D, concatenate

train_file = "total_train_data.csv"
data = pd.read_csv(train_file, delimiter=',')
queries = data['query'].tolist()
intents = data['intent'].tolist()

from util.Preprocess import Preprocess
p =
Preprocess(word2index_dic='../..train_tools/dict/chatbot_dict.bi
n',
            userdic='../..util/user_dic.tsv')
```

```
sequences = []
for sentence in queries:
    pos = p.pos(sentence)
    keywords = p.get_keywords(pos, without_tag=True)
    seq = p.get_wordidx_sequence(keywords)
    sequences.append(seq)

from config.GlobalParams import MAX_SEQ_LEN
padded_seqs = preprocessing.sequence.pad_sequences(sequences,
maxlen=MAX_SEQ_LEN, padding='post')

print(padded_seqs.shape)
print(len(intents))

ds = tf.data.Dataset.from_tensor_slices((padded_seqs, intents))
ds = ds.shuffle(len(queries))

train_size = int(len(padded_seqs) * 0.7)
val_size = int(len(padded_seqs) * 0.2)
test_size = int(len(padded_seqs) * 0.1)
```

```
train_ds = ds.take(train_size).batch(20)
val_ds = ds.skip(train_size).take(val_size).batch(20)
test_ds = ds.skip(train_size + val_size).take(test_size).batch(20)

dropout_prob = 0.5
EMB_SIZE = 128
EPOCH = 5
VOCAB_SIZE = len(p.word_index) + 1

input_layer = Input(shape=(MAX_SEQ_LEN,))
embedding_layer = Embedding(VOCAB_SIZE, EMB_SIZE,
input_length=MAX_SEQ_LEN)(input_layer)
dropout_emb = Dropout(rate=dropout_prob)(embedding_layer)

conv1 = Conv1D(
    filters=128,
    kernel_size=3,
    padding='valid',
    activation=tf.nn.relu)(dropout_emb)
pool1 = GlobalMaxPool1D()(conv1)
```

```
conv2 = Conv1D(  
    filters=128,  
    kernel_size=4,  
    padding='valid',  
    activation=tf.nn.relu)(dropout_emb)  
pool2 = GlobalMaxPool1D()(conv2)  
  
conv3 = Conv1D(  
    filters=128,  
    kernel_size=5,  
    padding='valid',  
    activation=tf.nn.relu)(dropout_emb)  
pool3 = GlobalMaxPool1D()(conv3)  
  
concat = concatenate([pool1, pool2, pool3])  
  
hidden = Dense(128, activation=tf.nn.relu)(concat)  
dropout_hidden = Dropout(rate=dropout_prob)(hidden)  
logits = Dense(5, name='logits')(dropout_hidden)  
predictions = Dense(5, activation=tf.nn.softmax)(logits)
```

```
model = Model(inputs=input_layer, outputs=predictions)
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(train_ds, validation_data=val_ds, epochs=EPOCH,
          verbose=1)

loss, accuracy = model.evaluate(test_ds, verbose=1)
print('Accuracy: %f' % (accuracy * 100))
print('loss: %f' % (loss))

model.save('intent_model.h5')
```

```
2023-05-28 05:52:01.401925: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1102] Device interconnect StreamExecutor with strength 1
2023-05-28 05:52:01.402001: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1108]
Epoch 1/5
3698/3698 [=====] - 42s 11ms/step - loss: 0.0482 - accuracy: 0.9843 - val_loss: 0.0169 - val_accuracy: 0.9948
Epoch 2/5
3698/3698 [=====] - 41s 11ms/step - loss: 0.0159 - accuracy: 0.9947 - val_loss: 0.0090 - val_accuracy: 0.9964
Epoch 3/5
3698/3698 [=====] - 41s 11ms/step - loss: 0.0116 - accuracy: 0.9958 - val_loss: 0.0082 - val_accuracy: 0.9967
Epoch 4/5
3698/3698 [=====] - 41s 11ms/step - loss: 0.0091 - accuracy: 0.9963 - val_loss: 0.0056 - val_accuracy: 0.9972
Epoch 5/5
3698/3698 [=====] - 41s 11ms/step - loss: 0.0081 - accuracy: 0.9966 - val_loss: 0.0057 - val_accuracy: 0.9975
529/529 [=====] - 0s 729us/step - loss: 0.0063 - accuracy: 0.9974
Accuracy: 99.744439
loss: 0.006294
```

CNN 모델을 이용하여 총 5회 학습을 진행한다.
학습을 통해 정확도를 99.7% 까지 올릴 수 있었으며,

이 과정을 통해 학습이 완료된 모델을 intent_model.h5라는 파일로 저장한다.

IntentModel.py

models/intent 폴더 안에 코드 작성

챗봇 엔진의 의도 분류 모듈을 생성. 앞서 학습한 의도 분류 모델 파일을 활용하여 입력되는 텍스트의 의도 클래스를 예측하는 기능을 구현한다.

```
import tensorflow as tf
from tensorflow.keras.models import Model, load_model
from tensorflow.keras import preprocessing

class IntentModel:
    def __init__(self, model_name, preprocess):

        self.labels = {0: "인사", 1: "욕설", 2: "주문", 3:
"예약", 4: "기타"}

        self.model = load_model(model_name)

        self.p = preprocess
```

```
def predict_class(self, query):  
    pos = self.p.pos(query)  
  
    keywords = self.p.get_keywords(pos, without_tag=True)  
    sequences = [self.p.get_wordidx_sequence(keywords)]  
  
    from config.GlobalParams import MAX_SEQ_LEN  
  
    padded_seqs =  
    preprocessing.sequence.pad_sequences(sequences,  
    maxlen=MAX_SEQ_LEN, padding='post')  
  
    predict = self.model.predict(padded_seqs)  
    predict_class = tf.math.argmax(predict, axis=1)  
    return predict_class.numpy()[0]
```

해당 코드는 엔진의 기능을 구현한 모듈의 기능만 하기 때문에, 이를 테스트 하기 위한 추가 코드 작성 필요.

model_intent_test.py

test 폴더 안에 코드 작성

```
import sys
sys.path.append('../')
from util.Preprocess import Preprocess
from models.intent.IntentModel import IntentModel

p = Preprocess(word2index_dic='../train_tools/dict/chatbot_dict.bin',
               userdic='../util/user_dic.tsv')

intent = IntentModel(model_name='../models/intent/intent_model.h5',
                    proprocess=p)
query = "오늘 탕수육 주문 가능한가요?"
predict = intent.predict_class(query)
predict_label = intent.labels[predict]

print(query)
print("의도 예측 클래스 : ", predict)
print("의도 예측 레이블 : ", predict_label)
```

```
2023-05-28 06:15:15.750032: I ter  
2023-05-28 06:15:15.750243: I ter  
2023-05-28 06:15:15.750321: I ter  
오늘 탕수육 주문 가능한가요?  
의도 예측 클래스 : 2  
의도 예측 레이블 : 주문
```

앞의 모델의 학습 결과 정확도가 99% 이상이 나왔기 때문에
학습 데이터에서 크게 벗어나지 않는 문장의 경우 의도를 잘 예측해서 출력한다.

개체명 인식 모델 학습

챗봇 엔진에 입력된 문장의 의도가 분류된 후, 문장 내 개체명 인식을 진행한다.
개체명 인식을 위해 양방향 LSTM이라는 모델을 사용하게 된다.
이번 모델을 구현할 때, 인식이 가능한 주요 개체명은 다음과 같다.

개체명	설명
B_FOOD	음식
B_DT, B_TI	날짜, 시간
B_PS	사람
B_OG	조직, 회사
B_LC	지역

train_model.py

models/ner 폴더 안에 코드 작성

버전 충돌로 인해 다운그레이드 필요.

pip uninstall matplotlib / pip uninstall numpy

pip install matplotlib==3.6 / pip install numpy==1.19.5

```
import sys
sys.path.append('../..')
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import preprocessing
from sklearn.model_selection import train_test_split
import numpy as np
from util.Preprocess import Preprocess
```

```
def read_file(file_name):  
    sents = []  
    with open(file_name, 'r', encoding='utf-8') as f:  
        lines = f.readlines()  
        for idx, l in enumerate(lines):  
            if l[0] == ';' and lines[idx + 1][0] == '$':  
                this_sent = []  
            elif l[0] == '$' and lines[idx - 1][0] == ';':  
                continue  
            elif l[0] == '\n':  
                sents.append(this_sent)  
            else:  
                this_sent.append(tuple(l.split()))  
    return sents  
  
p =  
Preprocess(word2index_dic='../train_tools/dict/chatbot_dict.bi  
n',  
            userdic='../util/user_dic.tsv')
```

```
corpus = read_file('ner_train.txt')

sentences, tags = [], []
for t in corpus:
    tagged_sentence = []
    sentence, bio_tag = [], []
    for w in t:
        tagged_sentence.append((w[1], w[3]))
        sentence.append(w[1])
        bio_tag.append(w[3])

    sentences.append(sentence)
    tags.append(bio_tag)

print("샘플 크기 : \n", len(sentences))
print("0번 째 샘플 단어 시퀀스 : \n", sentences[0])
print("0번 째 샘플 bio 태그 : \n", tags[0])
print("샘플 단어 시퀀스 최대 길이 :", max(len(l) for l in sentences))
print("샘플 단어 시퀀스 평균 길이 :", (sum(map(len,
sentences))/len(sentences)))
```

```
tag_tokenizer = preprocessing.text.Tokenizer(lower=False)
tag_tokenizer.fit_on_texts(tags)

vocab_size = len(p.word_index) + 1
tag_size = len(tag_tokenizer.word_index) + 1
print("BIO 태그 사전 크기 :", tag_size)
print("단어 사전 크기 :", vocab_size)

x_train = [p.get_wordidx_sequence(sent) for sent in sentences]
y_train = tag_tokenizer.texts_to_sequences(tags)

index_to_ner = tag_tokenizer.index_word
index_to_ner[0] = 'PAD'

max_len = 40
x_train = preprocessing.sequence.pad_sequences(x_train,
padding='post', maxlen=max_len)
y_train = preprocessing.sequence.pad_sequences(y_train,
padding='post', maxlen=max_len)
```

```
x_train, x_test, y_train, y_test = train_test_split(x_train, y_train,
test_size=.2, random_state=1234)

y_train = tf.keras.utils.to_categorical(y_train, num_classes=tag_size)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=tag_size)

print("학습 샘플 시퀀스 형상 : ", x_train.shape)
print("학습 샘플 레이블 형상 : ", y_train.shape)
print("테스트 샘플 시퀀스 형상 : ", x_test.shape)
print("테스트 샘플 레이블 형상 : ", y_test.shape)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Embedding, Dense, TimeDistributed,
Dropout, Bidirectional
from tensorflow.keras.optimizers import Adam

model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=30,
input_length=max_len, mask_zero=True))
model.add(Bidirectional(LSTM(200, return_sequences=True, dropout=0.50,
recurrent_dropout=0)))
model.add(TimeDistributed(Dense(tag_size, activation='softmax')))
model.compile(loss='categorical_crossentropy', optimizer=Adam(0.01),
metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=128, epochs=10)
```



```
print("평가 결과 : ", model.evaluate(x_test, y_test)[1])  
model.save('ner_model.h5')
```

```
def sequences_to_tag(sequences):  
    result = []  
    for sequence in sequences:  
        temp = []  
        for pred in sequence:  
            pred_index = np.argmax(pred)  
            temp.append(index_to_ner[pred_index].replace("PAD", "O"))  
        result.append(temp)  
    return result
```

```
from sequeval.metrics import f1_score, classification_report
```

```
y_predicted = model.predict(x_test)  
pred_tags = sequences_to_tag(y_predicted)  
test_tags = sequences_to_tag(y_test)
```

```
print(classification_report(test_tags, pred_tags))  
print("F1-score: {:.1%}".format(f1_score(test_tags, pred_tags)))
```

	precision	recall	f1-score	support
NP	1.00	1.00	1.00	303
_	0.62	0.53	0.57	658
_DT	1.00	1.00	1.00	13683
_FOOD	1.00	1.00	1.00	11655
_LC	0.78	0.58	0.66	314
_OG	0.64	0.47	0.54	460
_PS	0.77	0.49	0.60	396
_TI	0.69	0.77	0.73	61
micro avg	0.98	0.97	0.97	27530
macro avg	0.81	0.73	0.76	27530
weighted avg	0.98	0.97	0.97	27530
F1-score: 97.3%				

각 개체명의 밀도, 재현율, 검증 점수 가 출력된다.

개체명 인식 모델 학습이 완료되면 ner_model.h5 모델 파일 생성

NerModel.py

models/ner 폴더 안에 코드 작성

개체명 인식에 필요한 기능을 구현한 개체명 인식 모듈 작성

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import Model, load_model
from tensorflow.keras import preprocessing

class NerModel:
    def __init__(self, model_name, preprocess):

        self.index_to_ner = {1: 'O', 2: 'B_DT', 3: 'B_FOOD', 4: 'I', 5:
'B_OG', 6: 'B_PS', 7: 'B_LC', 8: 'NNP', 9: 'B_TI', 0: 'PAD'}

        self.model = load_model(model_name)

        self.p = preprocess
```

```
def predict(self, query):
    pos = self.p.pos(query)

    keywords = self.p.get_keywords(pos, without_tag=True)
    sequences = [self.p.get_wordidx_sequence(keywords)]

    max_len = 40
    padded_seqs = preprocessing.sequence.pad_sequences(sequences,
padding="post", value=0, maxlen=max_len)

    predict = self.model.predict(np.array([padded_seqs[0]]))
    predict_class = tf.math.argmax(predict, axis=-1)

    tags = [self.index_to_ner[i] for i in predict_class.numpy()[0]]
    return list(zip(keywords, tags))

def predict_tags(self, query):
    pos = self.p.pos(query)

    keywords = self.p.get_keywords(pos, without_tag=True)
    sequences = [self.p.get_wordidx_sequence(keywords)]
```

```
max_len = 40
padded_seqs = preprocessing.sequence.pad_sequences(sequences,
padding="post", value=0, maxlen=max_len)

predict = self.model.predict(np.array([padded_seqs[0]]))
predict_class = tf.math.argmax(predict, axis=-1)

tags = []
for tag_idx in predict_class.numpy()[0]:
    if tag_idx == 1: continue
    tags.append(self.index_to_ner[tag_idx])

if len(tags) == 0: return None
return tags
```

해당 NerModel 클래스를 테스트 하기 위해서 추가 코드 작성 필요.

model_ner_test.py

test 폴더 안에 코드 작성

개체명 인식에 필요한 기능을 구현한 개체명 인식 모듈 작성

```
import sys
sys.path.append('../')
from util.Preprocess import Preprocess
from models.ner.NerModel import NerModel

p = Preprocess(word2index_dic='../train_tools/dict/chatbot_dict.bin',
               userdic='../util/user_dic.tsv')

ner = NerModel(model_name='../models/ner/ner_model.h5', preprocess=p)
query = '오늘 오전 13시 2분에 탕수육 주문 하고 싶어요'
predicts = ner.predict(query)
tags = ner.predict_tags(query)
print(predicts)
print(tags)
```

```
[('오늘', 'B_DT'), ('오전', 'B_DT'), ('13시', 'B_DT'), ('2분', 'B_DT'), ('탕수육', 'B_FOOD'), ('주문', 'O'), ('하', 'O'), ('싶', 'O')]  
['B_DT', 'B_DT', 'B_DT', 'B_DT', 'B_FOOD']
```

코드에 넣은 테스트 문장의 개체명이 출력된다.

학습 데이터와 유사한 유형의 문장을 입력해서 개체명을 잘 인식했지만,
데이터와 많이 다른 문장을 넣으면 개체명 인식 정확도가 떨어질 수 있음.

답변 검색

입력된 문장의 전처리, 의도 분류, 개체명 인식 과정을 거쳐 해석된 데이터를 기반으로

적절한 답변을 학습 DB로부터 검색하는 기능을 구현.

검색 기능은 네이버 같은 포털사이트의 검색엔진

Database.py

util 폴더에 작성

답변 검색 기능을 사용할 때, 데이터베이스 접근과 제어를 쉽게 할 수 있도록,
해당 모듈을 먼저 생성한다.

```
import pymysql
import pymysql.cursors
import logging

class Database:
    def __init__(self, host, user, password, db_name, charset='utf8'):
        self.host = host
        self.user = user
        self.password = password
        self.charset = charset
        self.db_name = db_name
        self.conn = None
```

DB 연결

```
def connect(self):  
    if self.conn != None:  
        return  
  
    self.conn = pymysql.connect(  
        host=self.host,  
        user=self.user,  
        password=self.password,  
        db=self.db_name,  
        charset=self.charset  
    )
```

DB 연결 닫기

```
def close(self):  
    if self.conn is None:  
        return  
  
    if not self.conn.open:  
        self.conn = None  
        return  
    self.conn.close()  
    self.conn = None
```

SQL 구문 실행

```
def execute(self, sql):  
    last_row_id = -1  
    try:  
        with self.conn.cursor() as cursor:  
            cursor.execute(sql)  
            self.conn.commit()  
            last_row_id = cursor.lastrowid  
            # logging.debug("excute last_row_id : %d", last_row_id)  
    except Exception as ex:  
        logging.error(ex)
```

```
finally:  
    return last_row_id
```

SELECT 구문 실행 후, 단 1개의 데이터 ROW만 불러옴

```
def select_one(self, sql):  
    result = None  
  
    try:  
        with self.conn.cursor(pymysql.cursors.DictCursor) as cursor:  
            cursor.execute(sql)  
            result = cursor.fetchone()  
    except Exception as ex:  
        logging.error(ex)  
  
    finally:  
        return result
```

```
# SELECT 구문 실행 후, 전체 데이터 ROW만 불러옴
def select_all(self, sql):
    result = None

    try:
        with self.conn.cursor(pymysql.cursors.DictCursor) as cursor:
            cursor.execute(sql)
            result = cursor.fetchall()
    except Exception as ex:
        logging.error(ex)

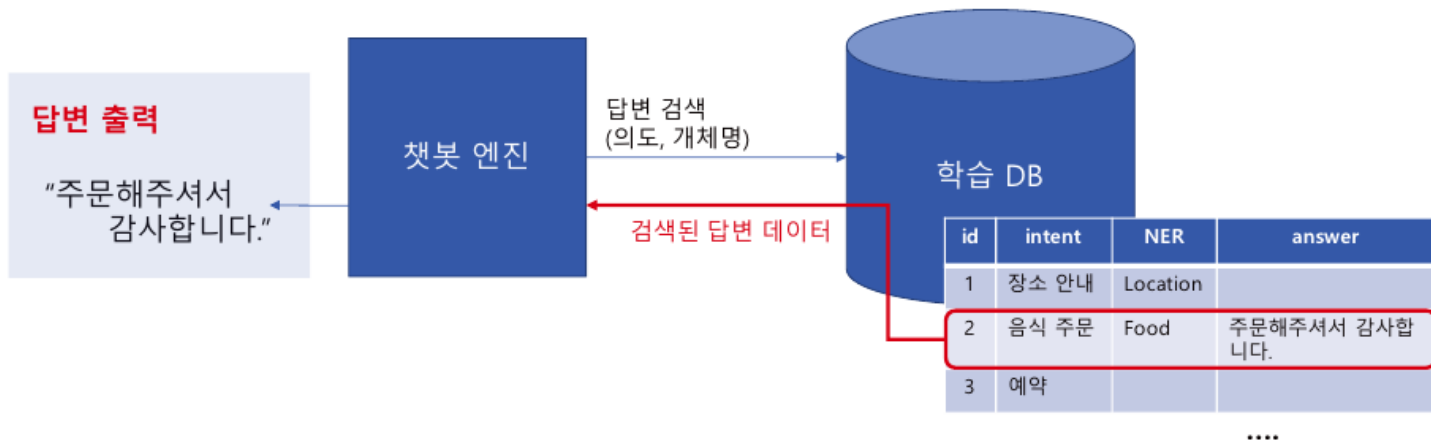
    finally:
        return result
```

해당 코드는 데이터베이스에 접근하고 전달받은 SQL문을 실행하는 제어 역할만 하기 때문에
검색 기능 구현을 위해서는 추가 모듈 구현 필요

답변 검색

전처리, 의도 분류, 개체명 인식 과정을 거쳐서 나온 자연어 해석 결과를 이용해 학습 DB에서 적절한 답변을 검색한다.

이 중에서 의도명과 개체명 2가지 항목을 가지고 답변을 검색하는 기능을 구현한다.



findAnswer.py

util 폴더에 작성

답변 검색에 필요한 기능을 제공하는 FindAnswer클래스를 생성한다.

```
class FindAnswer:
    def __init__(self, db):
        self.db = db

    # 검색 쿼리 생성
    def _make_query(self, intent_name, ner_tags):
        sql = "select * from chatbot_train_data"
        if intent_name != None and ner_tags == None:
            sql = sql + " where intent='{}' ".format(intent_name)

        elif intent_name != None and ner_tags != None:
            where = ' where intent="%s" ' % intent_name
            if (len(ner_tags) > 0):
                where += 'and ('
                for ne in ner_tags:
                    where += " ner like '{}{}%' or ".format(ne)
                where = where[:-3] + ')'
            sql = sql + where

    # 동일한 답변이 2개 이상인 경우, 랜덤으로 선택
    sql = sql + " order by rand() limit 1"
    return sql
```

답변 검색

```
def search(self, intent_name, ner_tags):
```

의도명, 개체명으로 답변 검색

```
    sql = self._make_query(intent_name, ner_tags)
```

```
    answer = self.db.select_one(sql)
```

검색되는 답변이 없으면 의도명만 검색

```
    if answer is None:
```

```
        sql = self._make_query(intent_name, None)
```

```
        answer = self.db.select_one(sql)
```

```
    return (answer['answer'], answer['answer_image'])
```

NER 태그를 실제 입력된 단어로 변환

```
def tag_to_word(self, ner_predicts, answer):
```

```
    for word, tag in ner_predicts:
```

변환해야하는 태그가 있는 경우 추가

```
        if tag == 'B_FOOD' or tag == 'B_DT' or tag == 'B_TT':
```

```
            answer = answer.replace(tag, word)
```

```
    answer = answer.replace('{', '')
```

```
    answer = answer.replace('}', '')
```

```
    return answer
```

chatbot_test.py

test 폴더에 작성

findAnswer로 구현한 검색 기능 클래스를 테스트 하는 코드를 작성
해당 코드가 입력받은 문장을 기반으로 답변을 찾아 출력하는 실질적인 챗봇의 엔진 역할을 한다.

```
import sys
sys.path.append('../')
from config.DatabaseConfig import *
from util.Database import Database
from util.Preprocess import Preprocess

# 전처리 객체 생성
p = Preprocess(word2index_dic='../train_tools/dict/chatbot_dict.bin',
               userdic='../util/user_dic.tsv')

# 질문/답변 학습 디비 연결 객체 생성
db = Database(
    host=DB_HOST, user=DB_USER, password=DB_PASSWORD, db_name=DB_NAME
)
db.connect()      # 디비 연결
```



```
query = "자장면 주문할게요"
```

```
# 의도 파악
```

```
from models.intent.IntentModel import IntentModel
intent = IntentModel(model_name='../models/intent/intent_model.h5',
proprocess=p)
predict = intent.predict_class(query)
intent_name = intent.labels[predict]
```

```
# 개체명 인식
```

```
from models.ner.NerModel import NerModel
ner = NerModel(model_name='../models/ner/ner_model.h5', proprocess=p)
predicts = ner.predict(query)
ner_tags = ner.predict_tags(query)
```

```
print("질문 : ", query)
print("=" * 100)
print("의도 파악 : ", intent_name)
print("개체명 인식 : ", predicts)
print("답변 검색에 필요한 NER 태그 : ", ner_tags)
print("=" * 100)
```

```

# 답변 검색
from util.findAnswer import FindAnswer

try:
    f = FindAnswer(db)
    answer_text, answer_image = f.search(intent_name, ner_tags)
    answer = f.tag_to_word(predicts, answer_text)
except:
    answer = "죄송해요 무슨 말인지 모르겠어요"

print("답변 : ", answer)

db.close() # 디비 연결 끊음

```

실행 시,
 질문, 답변 검색에 필요한
 의도, 개체명 추출.
 해당 질문에 맞는 답변을 검색하여 출력

질문 : 자장면 주문할게요

=====

의도 파악 : 주문

개체명 인식 : [('자장면', 'B_FOOD'), ('주문', 'O')]

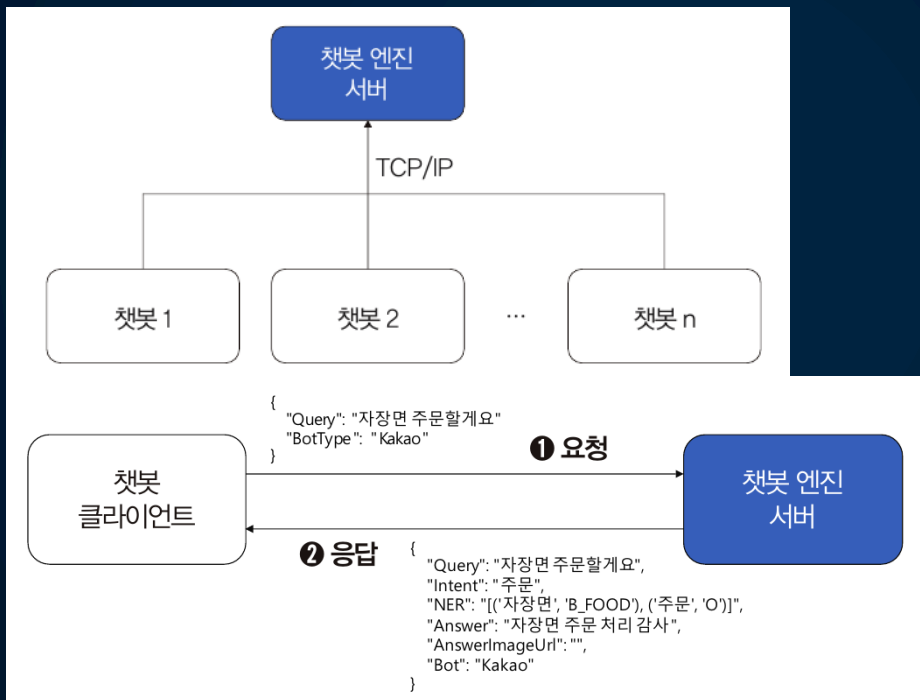
답변 검색에 필요한 NER 태그 : ['B_FOOD']

=====

답변 : 자장면 주문 처리 감사!!

서버 구현

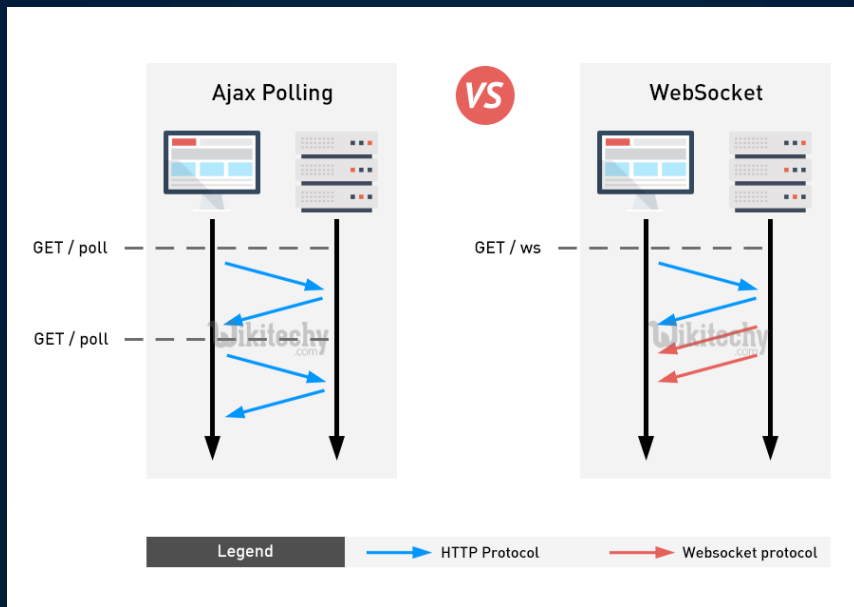
챗봇 엔진을 실제로 다양한 플랫폼에서 적용하기 위해 서버통신을 위한 기능을 구현.



소켓 서버

서버와 클라이언트가 특정 포트를 통해 실시간으로 양방향 통신을 하는 방식

HTTP통신 VS 소켓통신



BotServer.py

util 폴더에 작성

TCP 소켓 서버를 관리하는 모듈을 먼저 구현한다.

해당 모듈은 서버에 접속하는 클라이언트 소켓을 생성하고 처리하는 기능을 담당한다.

```
import socket

class BotServer:
    def __init__(self, srv_port, listen_num):
        self.port = srv_port
        self.listen = listen_num
        self.mySock = None

    def create_sock(self):
        self.mySock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.mySock.bind(("0.0.0.0", int(self.port)))
        self.mySock.listen(int(self.listen))
        return self.mySock

    def ready_for_client(self):
        return self.mySock.accept()

    def get_sock(self):
        return self.mySock
```

Bot.py

chatbot 폴더에 작성

최종적으로 실제 챗봇 기능을 담당하는 메인 프로그램을 구현한다.
챗봇 엔진 동작을 서버 환경에서 작동하기 위한 BotServer클래스,
여러 개의 클라이언트에서 동작될 수 있도록 멀티 스레드 모듈을 이용한다.

```
import threading
import json

from config.DatabaseConfig import *
from util.Database import Database
from util.BotServer import BotServer
from util.Preprocess import Preprocess
from models.intent.IntentModel import IntentModel
from models.ner.NerModel import NerModel
from util.findAnswer import FindAnswer

p = Preprocess(word2index_dic='train_tools/dict/chatbot_dict.bin',
               userdic='util/user_dic.tsv')

intent = IntentModel(model_name='models/intent/intent_model.h5', preprocess=p)

ner = NerModel(model_name='models/ner/ner_model.h5', preprocess=p)
```

```

def to_client(conn, addr, params):
    db = params['db']

    try:
        db.connect()  # 디비 연결

        # 데이터 수신
        read = conn.recv(2048)  # 수신 데이터가 있을 때 까지 블로킹
        print('=====')
        print('Connection from: %s' % str(addr))

        if read is None or not read:
            # 클라이언트 연결이 끊어지거나, 오류가 있는 경우
            print('클라이언트 연결 끊어짐')
            exit(0)

        # json 데이터로 변환
        recv_json_data = json.loads(read.decode())
        print("데이터 수신 : ", recv_json_data)
        query = recv_json_data['Query']

        # 의도 파악
        intent_predict = intent.predict_class(query)
        intent_name = intent.labels[intent_predict]

```

개체명 파악

```
ner_predicts = ner.predict(query)
ner_tags = ner.predict_tags(query)
```

답변 검색

try:

```
    f = FindAnswer(db)
    answer_text, answer_image = f.search(intent_name, ner_tags)
    answer = f.tag_to_word(ner_predicts, answer_text)
```

except:

```
    answer = "죄송해요 무슨 말인지 모르겠어요. 조금 더 공부 할게요."
    answer_image = None
```

```
send_json_data_str = {
    "Query" : query,
    "Answer": answer,
    "AnswerImageUrl" : answer_image,
    "Intent": intent_name,
    "NER": str(ner_predicts)
}
message = json.dumps(send_json_data_str)
conn.send(message.encode())
```



```
except Exception as ex:
    print(ex)

finally:
    if db is not None: # db 연결 끊기
        db.close()
    conn.close()

if __name__ == '__main__':

    # 질문/답변 학습 디비 연결 객체 생성
    db = Database(
        host=DB_HOST, user=DB_USER, password=DB_PASSWORD, db_name=DB_NAME
    )
    print("DB 접속")

    port = 5050
    listen = 100
```

봇 서버 동작

```
bot = BotServer(port, listen)
bot.create_sock()
print("bot start")

while True:
    conn, addr = bot.ready_for_client()
    params = {
        "db": db
    }

    client = threading.Thread(target=to_client, args=(
        conn,
        addr,
        params
    ))
    client.start()
```

해당 파일을 실행하면 서버가 활성화되고, 실행이 되는 중이라면 연결된 클라이언트에서 해당 챗봇의 기능을 사용할 수 있다.

DB 접속
bot start

chatbot_client_test.py

test 폴더에 작성

챗봇의 엔진을 담당하는 서버 프로그램을 구현했지만, 실제로 작동이 되는지를 확인하기 위해서 테스트 코드를 작성하여, 질문을 보냈을 때 그에 따른 적절한 답변이 나오는지 확인해본다.

```
import socket
import json

host = "127.0.0.1"
port = 5050

# 클라이언트 프로그램 시작
while True:
    print("질문 : ")
    query = input()
    if(query == "exit"):
        exit(0)
    print("-" * 40)

    # 챗봇 엔진 서버 연결
    mySocket = socket.socket()
    mySocket.connect((host, port))
```

챗봇 엔진 질의 요청

```
json_data = {  
    'Query': query,  
    'BotType': "MyService"  
}  
message = json.dumps(json_data)  
mySocket.send(message.encode())
```

챗봇 엔진 답변 출력

```
data = mySocket.recv(2048).decode()  
ret_data = json.loads(data)  
print("답변 : ")  
print(ret_data['Answer'])  
print("\n")
```

챗봇 엔진 서버 연결 소켓 닫기

```
mySocket.close()
```

클라이언트

질문 :
짜장면 배달 되나요

답변 :
짜장면 주문 처리 감사!!

서버

DB 접속
bot start

=====

```
Connection from: ('127.0.0.1', 13066)  
데이터 수신 : {'Query': '짜장면 배달 되나요', 'BotType': 'MyService'}
```