00. LangChain

▼ 개요
- 1. LangChain 훑어보기
- 2. LangChain 주요모듈
  - (1) 모델 I/O
  - (2) 데이터 연결
  - (3) 체인
  - (4) 메모리
  - (5) 에이전트/툴

▪ 1. LangChain 훑어보기

- 대규모 언어 모델(LLM)을 활용한 애플리케이션 개발에 특화된 오픈소스 프레임워크
- LLM을 잘 활용할 수 있도록 도와주는 모듈 모음

💡 랭체인에서 가능한 것

(1) 프롬프트,

▪ 2. LangChain 주요모듈

▫ (1) 모델 I/O

- 언어 모델과 상호작용을 위한 모듈
  - LLM에 전달될 프롬프트 생성
  - 답변을 받기 위해 API 호출
  - 답변에 대한 출력
- 프롬프트 : 입력 데이터와 검색 결과에 대한 것을 의미
- 언어 모델 : LLM 포함 채팅 모델, 임베딩 모델에 대한 API 호출 담당
- 파서 이용 : 구조화된 정보를 얻고 싶을 때 파서(Parsers) 이용
  - **PromptTemplate** : 프롬프트
  - **ChatOpenAI** : llm 모델
  - **HuggingFaceHub** : llm 모델
  - **ModelLaboratory** : 모델 성능 비교
  - **PydanticOuputParser** : (파서) 정의된 필드 타입에 맞게 자동 변환
  - **SimpleJosnOutputParser** : (파서) Json 형태 반환
  - **CommaSeparatedListOutputParser** : (파서) 콤마로 구분하여 결과 반환
  - **DatetimeOutputParser** : (파서) 날짜/시간 형태로 결과 반환
  - **XMLOutputParser** : (파서) XML 형태로 결과 반환

```
# 1) 프롬프트 생성
from langchain import PromptTemplate
template = "{product}를 홍보하기 위한 좋은 문구를 추천해줘"

prompt = PromptTemplate(
    input_variables=["product"],
    template=tempalte
)

prompt.format(product="카메라")
```

```
# 2) LLM 호출

## 2-1) openai
import os
os.environ['OPENAI_API_KEY'] = 'sk~'

from langchain.chat_models import ChatOpenAI
llm1 = ChatOpenAI(temperature=0, model_name="gpt-4o")
prompt = "진희는 강아지를 키우고 있습니다. 진희가 키우고 있는 동물은?"
print(llm1.predict(prompt)) # 왜 invoke가 아닌가


## 2-2) huggingface - falcon
import os
os.environ['HUGGINGFACEHUB_API_TOKEN'] = 'hf_'

from langchain import HuggingFaceHub
llm2 = HuggingFaceHub(repo_id = 'tiiuae/falcon-7b-instruct')
completion = llm2(prompt)
print(completion)


# 3) 모델 성능 비교
from langchian.model_laboratory import ModelLaboratory
model_lab = ModelLaboratory.from_llms([llm1, llm2])
model_lab.compare("대한민국의 가을은 몇 월부터 몇 월까지야?")
```

```
# 4) 출력 파서
from langchain.output_parsers import CommaSeparatedListOutputParser
from langchain.prompts import PromptTemplate

from langchain.chat_models import ChatOpenAI
llm = ChatOpenAI(temperature=0, max_tokens=2048, model_name='gpt-4o')

output_parser = CommaSeparatedListOutputParser() # 파서 초기화
format_instructions = output_parser.get_fomrat_instructions() # 출력 형식 지정

prompt = PromptTemplate(
```

```
    template='7개의 팀을 보여줘 {subject}.\n{format_instructions}',
    input_variables=['subject'],
    partial_variables={'format_instructions':format_instructions}
    )


query = "한국의 야구팀은?"
output = llm.predict(text=prompt.format(subject=query))

parsed_result = output_parser.parse(output)
print(parsed_result)
```

(2) 데이터 연결

- ETL : 데이터를 한 곳에서 다른 한 곳으로 옮기는 과정
  - 추출 (Extract) : 여러 출처로 부터 데이터를 가져옴
  - 변환 (Transform) : 추출한 데이터를 분석하고 필요한 형태로 변환
  - 적재 (Load) : 변환된 데이터를 데이터 베이스나 웨어하우스에 저장
- 데이터 연결 구성 요소
  - 문서 가져오기 : 다양한 출처에서 문서를 가져오는 것. ETL에서 추출에 해당
  - 문서 변환 : 입력 데이터를 청크로 분할 하거나 다시 결합하는 작업. ETL에서 변환에 해당
  - 문서 임베딩 : 데이터를 벡터로 변환
  - 벡터 저장소 : 변환된 벡터를 저장/관리/검색. ETL에서 적재
  - 검색기 : 언어 모델과 결합할 관련 문서를 가져오기 위한 것으로 정보 검색을 위한 역할
- RAG 실습 필요 라이브러리
  - pypdf : pdf 파일 다루는 데 사용
  - tiktoken : 임베딩을 하는 데 사용
  - faiss-cpu : 벡터 유사도 검색을 위해 사용
  - sentence-transofmrers : 문장 또는 단락을 벡터로 변환하기 위해 사용

```
# 1) 파일 로드
from langchain.document_loader import PyPDFLoader

loader = PyPDFLoader("e:/data/pdf.pdf")
document = loader.load()


# 2) 임베딩 처리
from langchain.vectorstores import FAISS
from langchain.embeddings import OpenAIEmbeddings

## 2-1) oepnai 임베딩
embeddings = OpenAIEmbeddings()
db = FAISS.from_documents(document, embeddings)

text = "진희는 강아지를 키우고 있습니다. 진희가 키우고 있는 동물은?"
text_embedding = embeddings.embed_query(text)

## 2-2) huggingface 임베딩
from langchain.embeddings import HuggingFaceEmbeddings
embeddings = HuggingFaceEmbeddings(model_name = "sentence-transform
text_embedding = embeddings.embed_query(text)


# 3) 검색기 활용
from langchain.chat_models import ChatOpenAI
llm = ChatOpenAI(temperature=0, model_name='gpt-4o')



qa = RetrievalQA.from_chain_type(
    llm = llm,
    chain_type="stuff",
    retriever =retriever)
    # chain_type
    # stuff : 모든 문서를 한 번에 붙여서 전달
    # map_reduce : 각각의 문서를 LLM에 넣어 부분적인 응답 생성 > 응답들을 LLM에 '
    # refine : 초기 답변 생성 후, 문서들을 하나씩 읽으면서 답변을 점진적으로 보완
```

```
    # map_rerank : 문서마다 개별적 응답 생성. score를 각각 부여하고 가장 적합한 응답

qeury = "마을 무덤에 있떤 남자를 죽인 사람은 누구니?"
result = qa({'query':query})
```

(3) 체인

- 여러 구성 요소를 조합해서 하나의 파이프라인을 구성해주는 역할
    - LLMChain
    - SequentialChain

```
from langchain.chains import LLMChain
from langchain import PromptTemplate
from langchain.chat_models import ChatOpenAI
import os

os.environ['OPENAI_API_kEY'] = 'sk~'
llm = ChatOpenAI(temperature=0, modle_name='gpt-4o')
prompt = PromptTemplate(input_variables=['country'])
template = "{country}의 수도는 어디야?'

chain = LLMChain(llm=llm, prompt=prompt)
chain.run('대한민국')
```

```
# chain1,
prompt1 = PromptTempalte(
    input_variables = ['sentence'],
    template = "다음 문장을 한글로 번역하세요. {sentence}"
)
chain1 = LLMChain(llm=llm, prompt=prompt1, output_key="translation")

# chain2,
prompt2 = PromptTempalte.from_template(
    "다음 문장을 한 문장으로 요약하세요. {translation}"
)
```

```
chain2 = LLMChain(llm=llm, prompt=prompt2, output_key="summary")

# allchain
from langchain.chains import SequentialChain
all_chain = SequentialChain(
    chains=[chain1, chain2],
    input_variables=['sentence'],
    output_variables=['translation','summary']
)

sentence = '''Over the past decade, social media platforms have transformed
all_chain(sentence)
```

▫ (4) 메모리

- 대화 과정에서 발생하는 데이터를 저장
  - 모든 대화 유지
  - 최근 K개의 대화 유지
  - 대화를 요약해서 유지
- ConversationChain 사용

```
from langchian.chat_modles import ChatOpenAI
llm = ChatOpenAI(temperature=0,
          model_name = 'gpt-4o'
          )

from langchain import ConversationChain
conversation = ConversationChain(llm=llm, verbose=True)

converstation.pedict(input="진희는 강아지를 한마리 키우고 있습니다.")
converstation.pedict(input="영수는 고양이를 두마리 키우고 있습니다.")
converstation.pedict(input="진희와 영수가 키우는 동물은 총 몇마리?")
```

▫ (5) 에이전트/툴

- 툴을 이용해서 특정 작업을 수행할 수 있는 에이전트\

```
# 에드시런 생년월일을 위키피디아에서 조회 > 계산기를 이용해 나이 계산
from langchain.chat_models import ChatOpenAI
llm = ChatOpenAI(temperature=0, model_name='gpt-4o')

from langchain.agents import load_tools
from langchain.agents import initialize_agent
from langchain.agents import AgentType

tools = load_tools(["wikipedia", "llm-math"], llm=llm)
# Tool Type
# python_repl : 파이썬 코드 실행 환경
# searx-serach : 오픈 소스 검색 엔진 serx 사용
# arxiv : arXiv 논문 검색 등
agent = initialize_agent(tools, # 에이전트가 접근할 수 있는 툴
        llm,
        agent = AgentType.ZERO_SHOT_REACT_DESCRIPTION, # 툴의 용도와 사용
        description='계산이 필요할 때',
        verbose=True)
agent.run("에드 시런이 태어난 해는? 2024년도 현재 에드  시런은 몇 살?")
# AgentType
# REACT_DOCSTORE : 질문에 답하기 위해 관련 정보 조회
# CONVERSATIONAL_REACT_DESCRIPTION : 메모리를 사용하여 과거에 시도했던
# SELF_ASK_WITH_SEARCH : 추가 질문을 스스로 만든 뒤, 검색 툴로 답을 찾아 종합
# OPENAI_MULTI_FUNCTIONS : funcion-calling 확장형. 복수 선택
```