졸업프로젝트 #1

축소된 지식기반 데이터베이스를 이용한 가벼운 한국어 KBQA 프레임워크 제안

건국대학교 컴퓨터공학부
201411310 장승원, 201410349 서원준, 201610099 이수민

0. 목차

1. 개요

트리플 데이터는 Subject-Predicate-Object 의 형태로 저장되는 오브젝트간의 관계를 나타내는 엔티티이다. 트리플 데이터는 기계가 이해할 수 있는 지식 (machine-readable knowledge) 이라는 측면에서 그 가치가 높지만 아직 크게 상용화되지 못했으며, 더욱이 영어가 아닌 한국어를 기반으로 하는 데이터셋 혹은 서비스는 찾아보기 힘들다. 이에 착안해 본 프로젝트에서는 Wikipedia 의 인포박스 데이터를 트리플 데이터로 저장하는 DBPedia 의 2016 년 한국어 덤프를 사용해, 색인이 가능한 한국어 트리플 데이터베이스를 구축하고, 일반 사용자도 쉽게 접근할 수 있도록 Simple Question 형태의 자연어 질의로 색인이 가능한 lightweight 프레임워크를 디자인한다.

2. 목표

1. Subject-Predicate-Object 형태의 자료를 담는 한국어 트리플 데이터베이스를 구축한다.
2. 요구 컴퓨팅 파워를 최소화한 KBDB 기반 Simple QA 프레임워크를 제안한다.
3. Simple Question 형태의 자연어 한국어 문장을 SPARQL 쿼리문으로 변환해 데이터베이스에 접근할 수 있도록 한다.
4. 입력 받은 자연어 질문에 대해 정답을 반환할 확률이 70% 이상이 되도록 한다.

3. 접근

1. 한국어 Wikipedia 의 인포박스 데이터를 트리플 형태로 저장하는 프로젝트 DBPedia 의 2016 년도 덤프의 일부를 내려받아 가공한 뒤, 축소된 데이터베이스로 활용한다. 또한 이 데이터베이스를 train-validate-test 로 분할해 인공지능 모델의 학습과 검증, 테스트에 사용한다.
2. Simple Question 은 기본적으로 문장 내에 Subject 와 Predicate 이 포함되어 있다. 자연어 처리 기법을 사용해 이 두 String 을 추출해낸다. Subject 추출에는 최근 주목받고 있는 BiLSTM 을 사용한 BIO 태깅 기법을 사용하되, 컴퓨팅 파워를 최대한 절약하기 위해 불필요한 부분을 제거하고 성능을 개선시키는 방향으로 접근하였다. 구축한 데이터베이스의 엔트리를 살펴본 결과, Subject 에 비해 종류가 현저히 적은 Predicate 를 추출하는 데에는 BiLSTM 이 아닌 CNN 을 사용하는 것으로 컴퓨팅 타임의 단축을 노렸다.

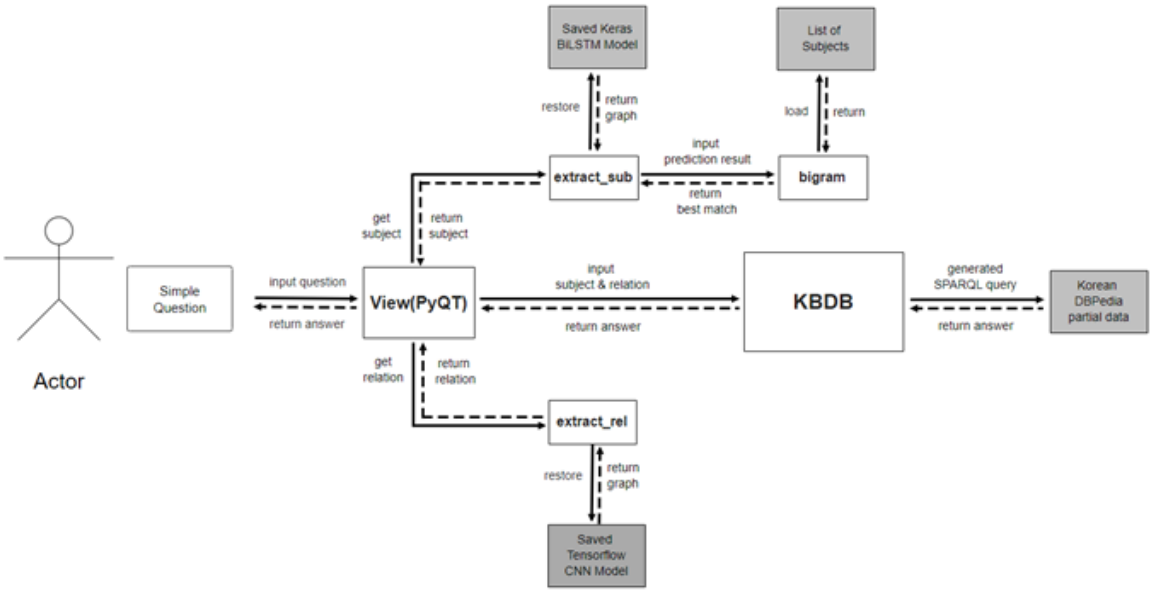
3. 추출해낸 Subject 와 Predicate 을 조합해 쿼리문을 생성한다. 트리플 데이터에 질의를 할 때는 SPARQL(Simple Protocol and RDF Query Language)을 사용하며, 파이썬 rdflib 라이브러리를 이용한다.
4. Subject 와 Predicate 이 올바르게 추출되면 SPARQL 질의의 정확도는 자연스럽게 따라오므로, Subject 추출과 Predicate 추출의 정확도를 올리는 데에 주력한다.

4. 요구사항 분석

1. 한국어 트리플 데이터베이스 구축
   A. 한국어 DBPedia 데이터 수집
      i. 시맨틱 트리플 데이터 형태의 한국어 데이터베이스 덤프를 수집해야 한다.
   B. 충분한 수의 데이터 확보
      i. 데이터 수집 후 가공 결과 집계된 트리플 데이터 엔트리의 수가 30 만개 이상이어야 한다.
      ii. 데이터 수집 후 가공 결과 집계된 Subject 와 Object 의 종류가 각각 10 만개 이상이어야 한다.
      iii. 데이터 수집 후 가공 결과 집계된 Predicate 의 종류가 200 개 이상이어야 한다.
2. 한국어 Simple Question 의 Sparql 변환
   A. 자연어 입력 문장의 SPARQL 변환을 위한 AI 모델 구축
      i. 입력 문장에서 Subject 를 추출하기 위해 Bi-LSTM(Bidirectional Long Short-Term Memory) 인공지능 모델을 구축 및 학습시켜야 한다.
      ii. 입력 문장에서 Predicate 을 추출하기 위해 CNN(Convolutional Neural Network) 인공지능 모델을 구축 및 학습시켜야 한다.
   B. 구축된 AI 모델을 통한 자연어 문장의 Subject 와 Predicate 추출
      i. 학습시킨 BiLSTM 모델과 CNN 모델을 사용해, 입력받은 한국어 Simple question 으로부터 Subject 와 Predicate 을 올바르게 추출해낼 수 있어야 한다.
   C. 추출된 Subject 와 Predicate 을 조합해 SPARQL 쿼리문 작성 및 질의
      i. Subject 와 Predicate 을 기반으로 Object 를 찾는 쿼리 스트링이 완성되어야 한다.
      ii. 작성된 SPARQL 을 사전 구축한 데이터베이스에 질의해 올바른 Object 값이 반환되어야 한다.
3. Simple Question 정확도 80% 이상 달성
   A. NL to SPARQL 과 SPARQL 질의 결과 정답률 향상
      i. 2019 년 전북대학교 논문인 [지식 베이스와 검색 기반 QA 의 결합 모델에 기반한 오픈 도메인 질의 응답]에서 제시한 관계 추출 (NL to SPARQL)의 정확도 85%, 정답 추출의 정확도(54%)를 상회하는 것을 목표로 한다.
      ii. NL to SPARQL 과 SPARQL 질의 결과의 정확도를 각각 80%까지 끌어올린다.

B. 달성하지 못할 경우 최적의 방법을 찾도록 한다.

5. 상위 디자인



<그림 1. 아키텍쳐 다이어그램>

디자인 해설

Actor

프로그램의 사용자이다. 사용자는 View 를 통해 나머지 모델과 상호작용할 수 있으며, 주요 액션은 Simple Question 을 집어넣고 그에 대한 answer 를 되돌려받는 것이다.

Simple Question

사용자가 입력하는 질문으로, 간단한 형식의 질문으로 제한한다. 이 프로젝트에서는 1 개의 트리플 데이터로 답을 도출 할 수 있는 질문을 Simple Question 으로 정의한다. 예를 들어 "이병헌의 나이는 몇 살인가?"에 대한 질문은 <이병헌>-<나이>-<49> 의 트리플 데이터로 처리가 가능하다. 하지만 "서울 시장의 나이는 몇 살인가?"와 같은 질문의 경우, <서울>-<시장>-<박원순>의 트리플 데이터와 <박원순>-<나이>-<65>의 데이터까지 총 2 개의 트리플 데이터가 필요하게 된다. 이러한 질문은 Simple Question 으로 분류하지 않기로 한다.

View

UI 를 입힌 MainController 이다. 최초 실행 시 하위 모델을 불러온 뒤, UI 를 실행하여 사용자와의 상호작용을 받는 Ready 상태가 된다. 하위 모델은 extract_sub, extract_rel, kbdb 클래스로 각각 입력한 문장으로부터 Subject 와 Predicate 를 추출하는 역할, 그리고 추출해낸 Subject 와 Predicate 에 대응하는 Object 를 데이터베이스에서 가져오는 역할을 한다.

Ready 상태의 모델은 TextBox 를 통해 Actor 로부터 Simple Question 을 입력받고, 질의 버튼을 누르면 이 Simple Question 을 하위 모델의 메소드에 인자로써 전달한다. extract_sub 과 extract_rel 클래스는 입력받은 문장으로부터 Subject 와 Predicate 데이터를 추출하고 View 에 반환하고, View 에서는 마지막으로 Subject 와 Predicate String 을 조합해 SPARQL 쿼리문을 생성, KBDB 에 질의를 넣게 된다. 마지막으로 KBDB 로부터 반환받은 최종 결과값은 사용자가 입력한 Simple Question 에 대한 대답이 되며, 이 반환값은 UI 의 답안 란에 표시된다.

extract_sub

입력받은 Simple Question 의 Subject 를 추출하는 BiLSTM 모델을 담고 있는 클래스이다. 우선 사전 train 을 마친 BiLSTM 모델은 파이썬의 pickle 패키지를 이용, /model/sub_model.pkl 에 저장되어 있다. 따라서 최초 기동시에는 pickle.load()명령으로 저장된 모델을 restore 하는 작업이 필요하다.

Subject 를 추출하는 과정에서는 BIO tagging 기법을 이용한 BiLSTM 모델을 사용한다. 영어 기반 자료인 기존의 BIO tagging 과정에서는 입력받은 Simple Question 을 띄어쓰기 단위로 분해하는데, 한국어의 경우 Subject 의 끝에 조사가 달려 있기 때문에 띄어쓰기 단위로 단어를 분해하기에는 단어 자체의 의미가 받는 손상이 크다. 이렇게 명사 뒤에 따라오는 조사를 제거하려면 형태소 분석기를 사용하는 것이 일반적이지만, 대표적인 파이썬의 한국어 형태소 분석 패키지인 konlpy.tag 를 사용해 본 결과, 고유명사를 잘 인식하지 못하고 올바르지 못하게 단어를 분해하는 경우가 상당히 많았다. 따라서 본 프로젝트에서는 이 과정에서의 데이터 손상을 최소화하기 위해, 어절 단위가 아닌 음절 단위로 BIO tagging 을 진행하기로 하였다.

BIO tagging 은 개체명 인식 기법으로, 자연어 문장에서 인식한 객체의 시작점은 Begin 을 뜻하는 'B'로, 객체의 일부이지만 시작점이 아닌 부분은 In 를 뜻하는 'I'로, 객체에 해당하지 않는 부분은 Out 을 뜻하는 'O'로 표기한다. 이를 이용해 Simple Question 의 주어를 'B', 'I'로 태깅하도록 모델을 학습시키고 실제 프로그램에 적용한다. 본 프로젝트의 경우 한국어 문장의 음절 단위 분해를 적용했기에, BiLSTM 모델은 예를 들자면 '해리포터는 어디에 사는가?'라는 문장을 ['B', 'I', 'I', 'I', 'O', 'O', 'O' ....] 형태로

변환할 것이다. 이 형태에서 'B'와 'I'에 해당하는 데이터만을 추출한 것이 우리가 원하는 Subject 가 된다.

이러한 과정을 통해 추출한 Subject 는 상당한 정확도를 보이지만, 데이터베이스에 질의를 넣기에는 아직 부족하다. 따라서 extract_sub 의 절차가 종료되면, 추출 과정에서 잘못된 결과를 반환했을 경우와, 사용자가 입력한 Subject 가 데이터베이스에 있는 엔트리와 유사한 이음동의어일 경우를 고려해, Bigram Feature 를 사용한 유사도 검사를 진행하기 위해 bigram 클래스에 이 결과를 넘겨준다.

bigram 에서 유사도 검사를 한 후 extract_sub 에서 다시 데이터를 받게 되는데 최종적으로 평가하기 위하여 전체 모델의 중심이자 mainclass 인 View 에 데이터를 넘김으로서 평가를 진행하도록 한다.

bigram

extract_sub 클래스에서 추출해낸 Subject 를 받아 데이터베이스에 존재하는 모든 Subject 엔트리와의 유사도 검사를 수행하는 클래스이다. 이 클래스에서는 사전 생성된 데이터인 모든 Subject 를 담은 리스트를 색인해, 인자로 받은 Subject 와 가장 유사한 형태의 Subject 를 반환한다. 따라서 search() 메소드를 통과한 결과값은 언제나 데이터베이스 상에 존재하는 엔트리를 가리킬 것이며, 이는 Subject 추출의 정확도를 상당한 폭으로 증가시킨다.

bigram feature 를 사용한 유사도 검사 방식은 다음과 같다. 예를 들어 '해리포터는 어디에 사는가?' 라는 문장에서 Subject 를 추출한 결과, 오류로 인해 데이터베이스에는 존재하지 않는 '해리포'라는 단어가 Subject 로 등장했다고 하자. bigram 클래스에서는 우선 이 Subject 를 순차적으로 두 글자씩 묶은 ['해리', '리포'] 라는 리스트를 생성한다. search() 메소드는 데이터베이스의 모든 Subject 를 이와 같은 방식으로 두 글자씩 묶은 뒤, ['해리', '리포'] 라는 리스트의 엔트리를 가장 많이 포함하는 Subject 가 우리가 정말로 찾는 Subject 일 것이라고 추측한다. 즉, ['해리', '리포', '포터'] 로 분해되는 '해리포터'라는 Subject 가 실제 정답임을 쉽게 알아낼 수 있다.

extract_sub 으로부터 전달받은 Subject 는 bigram.search() 메소드를 통과한 뒤 다시 extract_sub 으로 반환한다. 이렇게 bigram feature 를 이용한 유사도 검사를 통해 더 정확한 subject 를 얻을 수 있다.

Keras BiLSTM Model

extract_sub 을 위해 사전에 학습시켜 둔 Keras BiLSTM 모델이다. 파이썬의 pickle 패키지를 이용해 저장해 두었으며, extract_sub 클래스가 최초 호출될 때 pickle.load() 메소드를 통해 restore 된다.

List of Subjects

bigram 클래스에서 사용할 파이썬 리스트이다. 데이터베이스의 모든 Subject 와 Object 에 대해서 ['해리', '리포', '포터'] 의 형태로 분해된 데이터를 저장하고 있다. 컴퓨팅 타임을 줄이기 위해서 매번 코드를 통해 데이터 가공을 실행하는 것보다는, 마찬가지로 pickle 패키지를 이용해 최초 실행 시 사전 생성된 리스트를 restore 하는 것을 선택했다.

extract_rel

입력받은 Simple Question 으로부터 Predicate 를 추출하는 클래스이다. 초기에 Subject 와 마찬가지로 BiLSTM 을 사용하려고 했으나, 프로젝트의 주제를 생각했을 때 조금 더 가볍고 컴퓨팅 파워를 절감할 수 있는 방법을 고민하게 되었다. 따라서 축소된 데이터셋을 사용하기 때문에 Predicate 의 종류가 적은 본 프로젝트에는, BiLSTM 이 등장하기 이전에 사용되던 CNN 모델이 오히려 적합한 모델이라고 판단, Predicate 추출에 사용하였다.

우선 뉴럴 네트워크의 output nodes 에 데이터베이스에 존재하는 모든 Predicate 를 할당한 뒤, 자연어 문장을 입력하면 그 Predicate 에 해당하는 output node 에 가장 높은 가중치가 부여되도록 CNN 모델을 학습시켰다. 실 사용 시에는 이렇게 학습된 모델을 불러온 뒤, extract_sub 의 경우와 동일하게 어절 단위가 아닌 음절 단위로 embedding 을 진행, CNN 모델에 처리된 문장을 통과시키게 된다. 이후 predict()결과 가장 높은 가중치를 나타내는 노드를 구하고, 그 노드에 해당하는 Predicate 을 View 에 돌려준다.

Tensorflow CNN Model

CNN 은 tensorflow 에서 제공하는 save 방식이 있어서 저장과 restore 에 이를 채택하였다. Keras 에서 사용한 pickle 과는 다르게 100 번 학습마다의 데이터를 저장하였으며, Tensorflow 1.14 에서 제공하는 vocabulary processor 을 이용하여 word Embedding 된 데이터를 저장하도록 하였다. 또한, NLP 된 데이터로 단어를 역추적하기 위한 데이터로 pickle 을 사용하여 각 클래스 번호 당 class 를 출력하도록 하였다.

extract_rel 이 동작하기 위하여 모델을 불러오는 것은 이 모델을 불러오는 것이며, 이를 통해 학습된 데이터로 extract_rel 가 작동한다.

KBDB

extract_sub, extract_rel 클래스로부터 추출을 마친 Subject 와 Predicate 을 조합해 SPARQL 쿼리를 생성하고, 데이터베이스에 그 쿼리를 날려 Object 데이터를 얻어 오는 클래스이다. 데이터베이스로는 한국어 DBPedia 의 2016 년 덤프의 일부인 mappingbased_objects_ko.ttl 을 사용한다. 이 파일은 약 58 만 줄의 트리플 데이터 엔트리를 가지고 있으며, 10 만 종의 Subject & Object 와 약 250 종의 Predicate 가 사용되었다.

DBPedia 덤프는 ttl 파일 형태로 저장된 트리플 데이터셋을 제공한다. 관계형 데이터베이스 색인에 SQL(Structured Query Language)이 사용되듯이, 트리플 데이터베이스를 색인하는 데에는 SPARQL(Simple Protocol and RDF Query Language)쿼리문이 사용된다. 본 프로젝트에서는 단순히 String 의 조합으로 매우 간단한 SPARQL 쿼리문을 생성했으며, RDF 데이터 탐색을 돕는 파이썬 패키지인 rdflib 을 사용해 데이터베이스에 질의를 보내 답변을 얻을 수 있었다.

Korean DBPedia Partial Data

한국어 DBPedia 2016 년 덤프 데이터 중, mappingbased_objects_ko.ttl 파일에 해당한다. 이 파일을 rdflib 을 이용해 읽어들여 KBDB 클래스에서 질의를 보내게 된다. 총 58 만개의 트리플 데이터 엔트리를 보유하고 있으며, 10 만종의 Subject & Object 와 약 250 종의 Predicate 가 사용되었다.

삭제된 모델

Morph

기존 디자인은 형태소 분석을 통하여 Word Embedding 을 하며 Simple Question 의 특성을 이용하여 맨 앞에 나오는 명사를 Subject 로 뽑을 예정이었다. 그러나 한국어 형태소 분석기 모델 역시 정확도가 약 85%밖에 되지 않았으며 가장 큰 문제로 대부분의 Subject 는 고유 명사이지만 한국어 형태소 분석기에서 고유 명사의 정확도는 매우 떨어지는 모습을 볼 수 있었다. 예를 들어 "하정우가 태어난 곳은 어디인가?" 라는 질문을

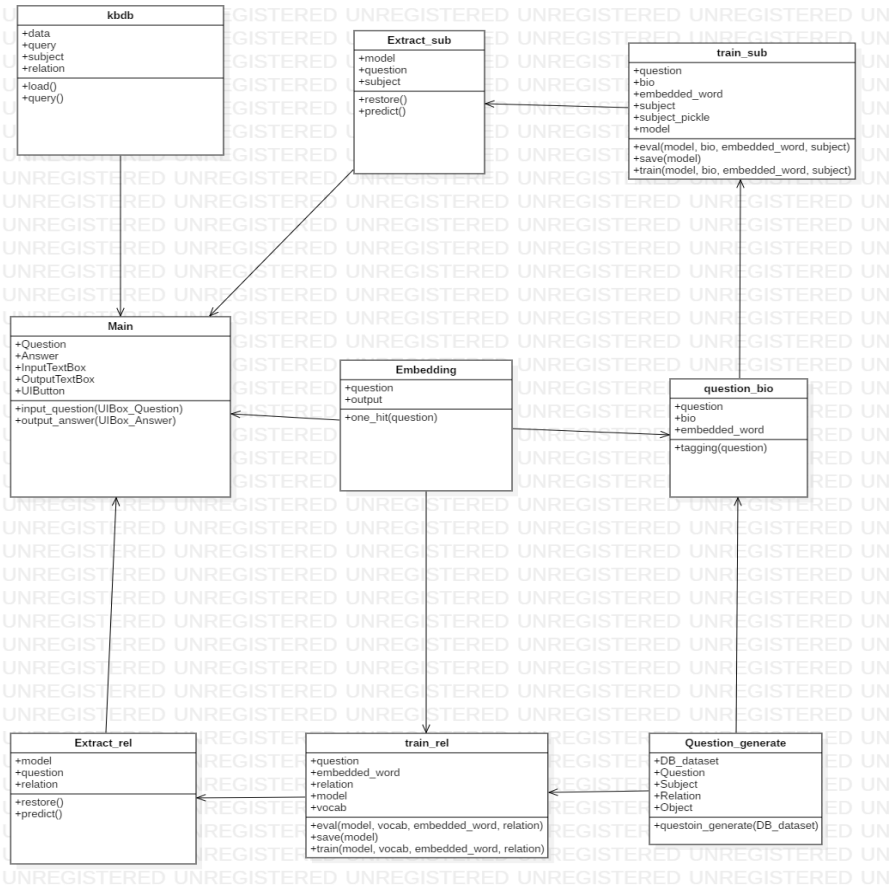
입력하게 되면 형태소 분석기 프로그램인 (KoNLPy)를 사용하게 되면 "하정/우/가/ / 태어/나/ㄴ/ /곳/은/ / 어디/인가/?"로 하정우라는 명사가 정확하게 나오지 않게 된다. 이런 오류 때문에 Word Embedding 을 형태소 분석이 아닌 음절 단위로 선택하게 되었으며 Morph 모델은 삭제하게 되었다.

Transformer

기존 Subject 를 추출하기 위한 방법으로 Transformer 를 이용하고자 하였다. 그러나 다른 논문들을 참조하니 Transformer 보다 BiLstm 을 사용하는 방식이 많았으며, BiLSTM 은 간편하게 사용할 수 있지만 Transformer 는 인코딩, 디코딩 방식과 Position Embedding 등 여러가지를 고려해야하기 때문에 컴퓨팅 파워 면으로 시간이 더 걸린다고 판단하였다.

6. 다이어그램

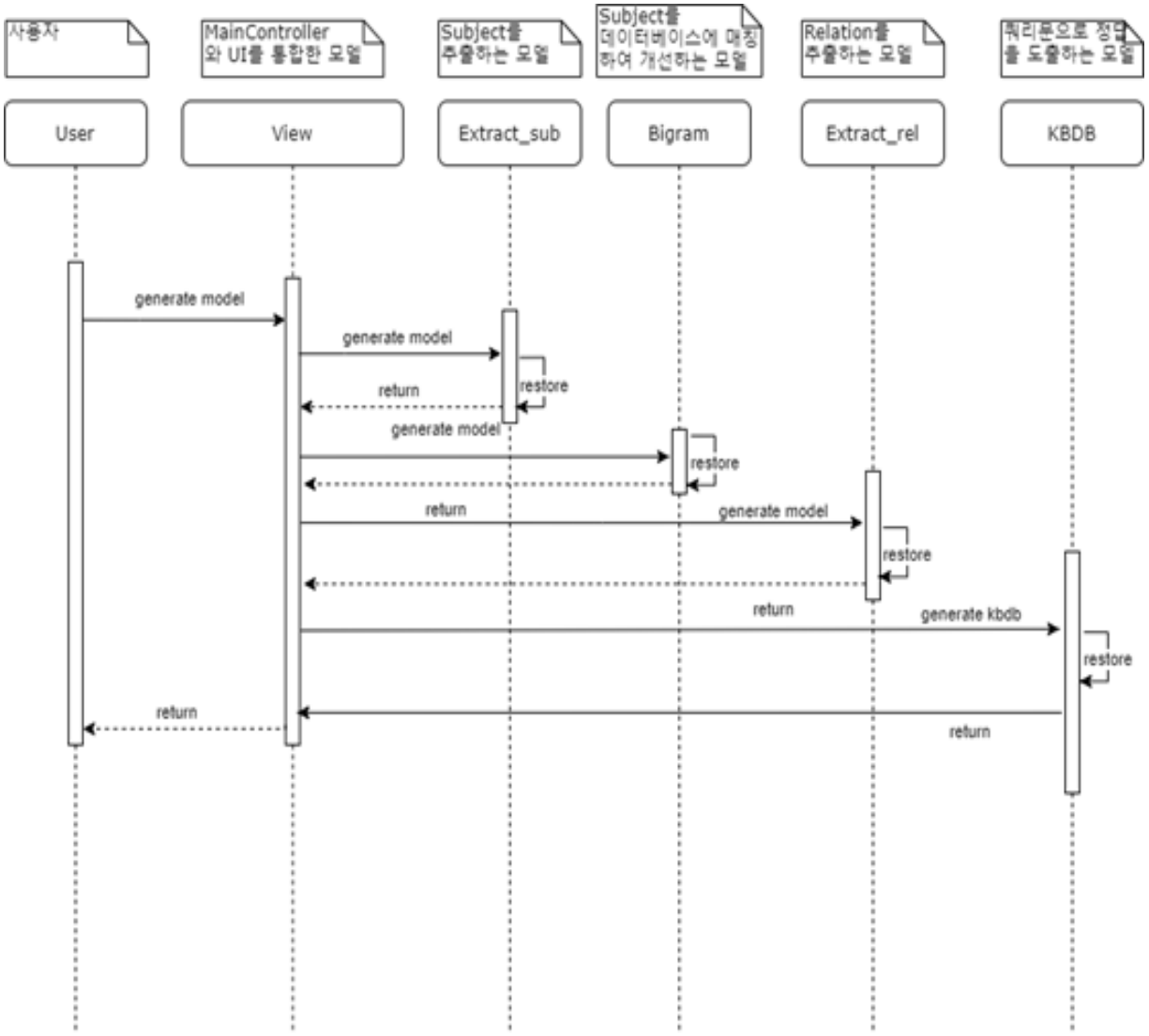
A. 클래스 다이어그램



<그림 2. 클래스 다이어그램>

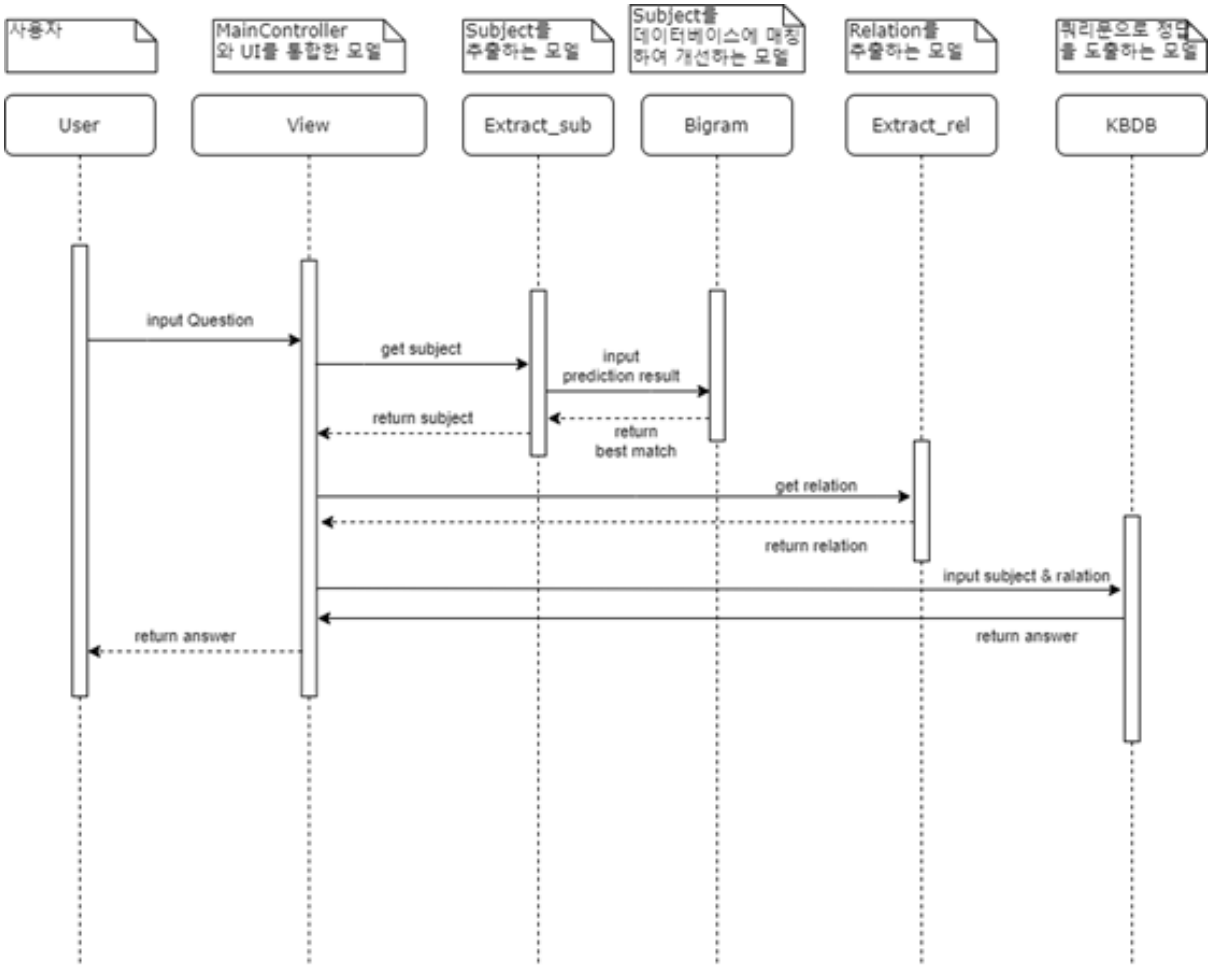
B. 시스템 시퀀스 다이어그램

i. 자연어 문장에서의 Subject 와 Predicate 추출



<그림 3. 시스템 시퀀스 다이어그램 (1)>

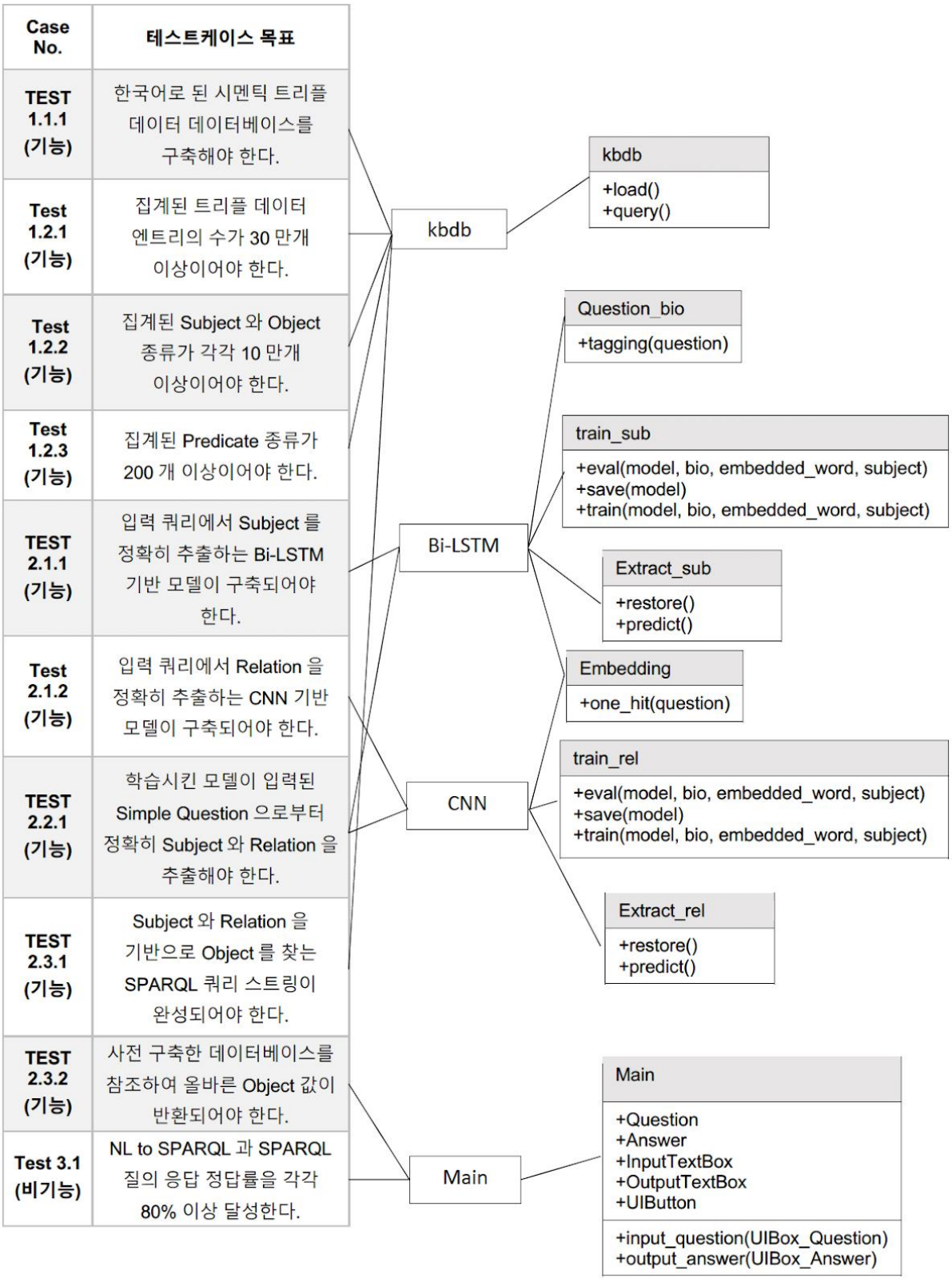
ii. 질의응답 시스템



<그림 4. 시스템 시퀀스 다이어그램 (2)>

C. 추적성 매트릭스

Case No.	테스트케이스 목표
TEST 1.1.1 (기능)	한국어로 된 시멘틱 트리플 데이터 데이터베이스를 구축해야 한다.
Test 1.2.1 (기능)	집계된 트리플 데이터 엔트리의 수가 30 만개 이상이어야 한다.
Test 1.2.2 (기능)	집계된 Subject 와 Object 종류가 각각 10 만개 이상이어야 한다.
Test 1.2.3 (기능)	집계된 Predicate 종류가 200 개 이상이어야 한다.
TEST 2.1.1 (기능)	입력 쿼리에서 Subject 를 정확히 추출하는 Bi-LSTM 기반 모델이 구축되어야 한다.
Test 2.1.2 (기능)	입력 쿼리에서 Relation 을 정확히 추출하는 CNN 기반 모델이 구축되어야 한다.
TEST 2.2.1 (기능)	학습시킨 모델이 입력된 Simple Question 으로부터 정확히 Subject 와 Relation 을 추출해야 한다.
TEST 2.3.1 (기능)	Subject 와 Relation 을 기반으로 Object 를 찾는 SPARQL 쿼리 스트링이 완성되어야 한다.
TEST 2.3.2 (기능)	사전 구축한 데이터베이스를 참조하여 올바른 Object 값이 반환되어야 한다.
Test 3.1 (비기능)	NL to SPARQL 과 SPARQL 질의 응답 정답률을 각각 80% 이상 달성한다.



<그림 5. 추적성 매트릭스>

7. 진행 과정

A. 한국어 DBPedia 질문 데이터 생성

앞서 프로젝트 개요에서 언급했듯이, 한국어로 된 트리플 형식의 데이터는 찾아보기 힘들며, 존재한다 하여도 도서나 논문, 특허 등 데이터가 다루는 범위가 다소 제한적이다. 따라서 보다 광범위한 분야에 대한 데이터를 얻기 위해 Wikipedia 인포박스 데이터를 트리플 데이터로 저장한 DBPedia 의 2016 년도 한국어 덤프 데이터의 일부를 사용하였다.

본 프로젝트는 사용자가 입력한 질문을 입력값으로 받는다. 해당 입력값은 사람들이 일상 생활에서 사용하는 자연어(Natural Language)이므로, 이를 트리플 데이터 형식으로 변환하기 위해서는 질문에서 Subject 와 Predicate 이 무엇인지 알아낼 수 있어야 한다. 이 과정은 각각 Bi-LSTM 과 CNN 알고리즘을 기반으로 도출되며, 이 과정이 성공적으로 진행되었음을 전제로 트리플 데이터 데이터베이스에서 Object 를 추출한다. 따라서 연구 진행을 위하여 한국어 트리플 데이터로 구성된 데이터베이스가 필요한데, 이를 위해 한국어 DBPedia 데이터의 정제 과정이 요구되었다.

데이터셋 구축 및 정제 과정은 다음과 같다. DBPedia 덤프 데이터의 Subject, Predicate, Object 데이터는 모두 다음과 같이 인터넷의 자원의 주소인 URI(Uniform Resource Identifier) 형식을 따른다.

```
<http://ko.dbpedia.org/resource/야나미_조지> <http://dbpedia.org/ontology/birthPlace> <http://ko.dbpedia.org/resource/도쿄_시> .
<http://ko.dbpedia.org/resource/야나미_조지> <http://xmlns.com/foaf/0.1/homepage> <http://www.aoni.co.jp/actor/ya/yanami-jouji.html> .
<http://ko.dbpedia.org/resource/일요일의_손님들> <http://dbpedia.org/ontology/director> <http://ko.dbpedia.org/resource/김수용_(영화_감독)> .
<http://ko.dbpedia.org/resource/일요일의_손님들> <http://dbpedia.org/ontology/country> <http://ko.dbpedia.org/resource/South_Korea> .
<http://ko.dbpedia.org/resource/일요일의_손님들> <http://dbpedia.org/ontology/language> <http://ko.dbpedia.org/resource/한국어> .
```

<그림 6. 한국어 DBPedia 데이터셋의 일부>

데이터를 학습시키는 과정에서 URI 데이터 중 자연어가 아닌 부분은 무의미하기에 자연어 이외의 부분은 제외해야 했고, pandas 라이브러리와 rdflib 라이브러리를 이용하여 다음과 같이 오직 자연어로만 구성된 트리플 데이터셋으로 변환했다. Subject 데이터 중 괄호나 특수기호가 포함된 데이터는 괄호 안의 내용과 특수기호를 제거하였다.

```
야나미_조지,birthPlace,도쿄_시
벤다,leaderName,파트릭_음페푸
아메리칸_하키_리그,country,Canada
이파리과,family,이파리상과
왓_위민_원트,musicComposer,앨런_실베스트리
```

<그림 7. URI 형태에서 String 형태로 가공한 DBPedia 데이터셋의 일부>

또한 학습의 관점에서 사용자가 실제로 어떻게 질문을 하는지에 대한 질문 데이터셋 구축이 필요하였기에 Predicate 에 대한 질문 데이터를 수기로 작성하였다. 질문 데이터는 실제 사람이 질문하는 어투와 유사하도록 하는 것을 목표로 하였기에 받침 유무에 따라 조사를 고려하여 생성되도록 하였고, 어미 또한 다양하게 작성하였다. 본 프로젝트가 사용한 덤프 파일은 'mappingbased_objects_ko.ttl' 파일이며, 해당 파일에 존재하는 도합 242 개의 Predicate 에 대한 질문 양식을 각 Predicate 당 적게는 5 개, 많게는 10 개의 질문을 작성하였으며 최종적으로 약 2000 개의 질문 데이터를 생성하였다. 최종으로 생성된 데이터셋은 다음과 같다.

```
K-pop_Selection,genre,K-pop,K-pop_Selection의 장르가 뭐예요?
FROM_ME_TO_YOU,genre,J-pop,FROM_ME_TO_YOU의 장르는 뭐야?
쌩뚱맞고,developer,엠게임,쌩뚱맞고의 개발사는 어디인가?
소스포지_엔터프라이즈_에디션,developer,콜랩넷,소스포지_엔터프라이즈_에디션의 개발사는 어디입니까?
점프_얼티밋_스타즈,developer,간바리온,점프_얼티밋_스타즈의 개발사는 어디야?
스포어,developer,맥시스,스포어의 개발사는 어디인가요?
```

<그림 8. Simple Question 데이터를 추가한 DBPedia 데이터셋>

실행 절차

1. DBPedia 한국어 버전으로부터 Triple(Subject, Predicate, Object) 데이터를 확보
   A. 2016 년도 DBPedia 데이터를 dump, 그 중 일부를 사용
   B. 데이터셋 중, 괄호나 특수기호가 포함된 데이터는 그 특수기호를 삭제
   C. triple 구조 데이터를 pandas 의 dataFrame 구조로 추출한다.
      i. Subject - <http://ko.dbpedia.org/resource/야나미_조지>
      ii. Predicate - <http://dbpedia.org/ontology/birthPlace>
      iii. Object - <http://ko.dbpedia.org/resource/도쿄_시>
2. 수집한 데이터를 기반으로 Simple Question 데이터 작성
   A. 실제 사람이 질문하는 어투와 유사하도록 조사와 어미 등을 고려하여, Predicate 하나 당 여러 형태의 질문을 작성
   B. 정확도 개선을 위하여 더 많은 Predicate 확보 및 그에 따른 Question Dataset 추가 생성

B. Bilstm 과 BIO tagging 을 이용한 Subject 추출

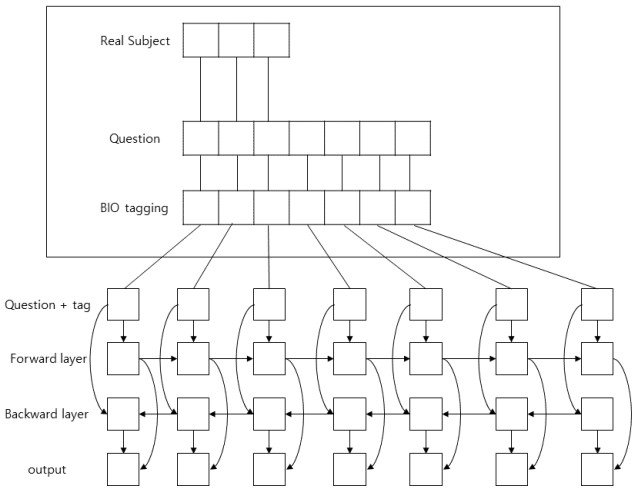
개체명 인식을 하기 위해서는 일반적으로 BiLSTM 인공지능 모델이 사용된다. 본 프로젝트의 경우 개체명 중에서도 Subject 의 추출이 필요한데, BIO tagging 방식을 이용하여 더욱 간편히 Subject 를 추출하도록 한다. 기존 NLP 의 BIO tagging 은 문장을 space 단위로 분해하는 것이 일반적이지만, 본 프로젝트의 경우 한국어를 다루기 때문에, 주어에 딸려오는 조사 등의 복잡성을 고려해 음절 단위로 분해하기로 한다.

분해된 문장에서 BIO tagging 을 이용해 Subject 에 'B'와 'I'를 맵핑한 후에는, 이 리스트를 인공지능 모델에 투입하기 위해 리스트의 길이를 통일시켜 주어야 한다. 따라서 사전에 정의해 둔 MAX_LENGTH 상수를 이용해, 모든 리스트가 같은 길이가 될 때까지 뒤에 0 을 채워넣는 방식으로 확장한다.

실행 절차

1. Question BIO 처리
   A. 자연어 문장을 음절 단위의 리스트로 분해
   B. Subject 에 해당하는 부분을 'B', 'I', 'O'로 인코딩
      i. 시작 부분을 Begin 을 뜻하는 'B'로 인코딩
      ii. 시작 부분이 아닌 Subject 의 나머지 부분을 In 을 뜻하는 'I'로 인코딩
      iii. Subject 에 해당하지 않는 부분을 Out 을 뜻하는 'O'로 인코딩
   C. 리스트의 길이가 MAX_LENGTH 상수와 같아질 때까지 Zero-Padding 을 진행한다.

2. BIO tagging 이 완료된 문장 반환
   A. Pad 된 데이터는 output 문장에서 고려하지 않음
   B. Input - 김대중이 태어난 곳은 어디인가?
   C. output = [김, B], [대, I], [중, I], [이, O], [태, O], ... [가, O], [?, O]



<그림 9. 본 프로젝트에서 사용된 Bi-LSTM 모델의 레이아웃>

1. keras sequential model 내의 Bi-LSTM 을 이용하여 모델을 구축한다.
   A. 주어에 해당하는 부분의 객체명을 인식하도록 학습된 모델을 사용
   B. 입력받은 음절은 자주 등장하는 순으로 1 부터 정수를 매겨 토큰화
   C. 글자 당 BIO 로 tag 된 데이터가 출력
   D. BIO 태깅에서 B, I 만 선택하여 Predicted Subject 를 생성

2. Subject 가 정확히 추출되었는지 평가한다.
   A. BIO 의 tagging 방식이 얼마나 정확한지를 확인한다. 또한, 그 데이터들을 비교 분석하여 B 의 정확도, I 의 정확도, O 의 정확도를 파악하여 어느 부분을 개선해야 하는지 확인한다.
   B. 생성된 단어와 실제 입력 단어를 비교한다. 이 때의 정확도를 predict Subject 의 정확도로 본다.

C. CNN 을 이용한 Predicate 추출

축소된 데이터셋을 사용하는 본 프로젝트에서 사용되는 Subject 의 종류는 약 10 만 개를 넘어서지만, Predicate 의 경우 약 250 종이 등장한다. 따라서 컴퓨팅 파워가 비교적 많이 소모되며 처리시간이 긴 Bi-LSTM 을 사용하는 접근보다는, 단순 Classification 문제로 취급해 CNN 을 사용하는 것이 적합하다는 결론에 이르렀다.

실행 절차

1. DBPedia Predicate 정보 토큰화
    A. Predicate 의 경우 그 종류가 매우 적으므로 출현 빈도와 관계없이 dataset 에 등장하는 순서대로 토큰화
2. CNN 을 이용하여 입력된 문장 Predicate 으로 추출
    A. 전체 Predicate 의 종류만큼의 Output Nodes 를 가진 CNN 을 사용
        i. Subject 와 마찬가지로 어절 단위로 분해한 문장을 입력
        ii. Output Nodes 중 가장 높은 가중치를 보이는 노드 번호 획득
        iii. 사전 생성한 토큰화 데이터를 이용해 해당 정수를 자연어 단어로 변환

D. 정확도 분석

요구사항에서 기술한 정확도를 위하여, 전체 NL to SPARQL 단계의 정확도를 알아보고자 한다. 이를 위해 이전 단계에서 추출한 Subject 와 Predicate 의 정확도를 측정하고 이를 종합적으로 평가한다.

실행 절차

1. Subject 의 정확도 측정
    A. 학습 데이터셋에서 사용자가 입력하지 않을 만한 특수기호를 모두 제거
    B. 학습 데이터셋을 80:20 로 나누어 각각 train & validate 데이터로 사용
    C. 단순한 BIO 태깅 결과 정확도는 91%
        i. but! 대부분 O 의 정확도이다
        ii. 실제 B 또는 I 로 태깅 되는 데이터는 전체 문장의 15%정도
    D. 하지만 과도하거나 부족한 'B', 'I' 태깅으로 Subject 의 정확도는 35.8%
        i. Question input - 조지 워싱턴은 어디서 태어났는가?
        ii. Subject input - 조지 워싱턴
        iii. Predict Subject – [조지 워] or [조지 워싱턴은]

2. Predicate 의 정확도 측정
   A. 학습 데이터셋에서 사용자가 입력하지 않을 만한 특수기호를 모두 제거
   B. 학습 데이터셋을 85:15 로 나누어 각각 train & validate 데이터로 사용
   C. Predicate 의 정확도는 76.8%

3. 문제점 및 개선방안
   A. DBPedia 엔트리의 모든 space 는 언더바(_)로 대체되어 있음
      i. 주로 subject 의 정확도가 높지 않은 이유가 이 곳에서도 발생하였다. DBPedia 데이터에서 우리가 제거한 특수문장은 띄어 쓰기를 _로 바꾸는 것 이외에도 세부사항(예를 들어, 괄호 내부의 데이터가 삭제가 된 것이 많다.)들이 제거가 되었기 때문에 35.8%의 정확도로 찾아도 틀린 경우도 다수 발생하였다.
   B. subject 의 문제점으로 BI tag 가 정확하지 않은 경우를 보이게 된다. 특히 B 가 문장 속에 1 개만 있어야 하지만 여러개 발생하는 경우와, 정확도에 영향을 미치는 I 로 tag 된 개수가 차이가 심한 경우를 볼 수 있다.
   C. Predicate 의 가장 큰 문제점으로 선택된 것은 Word Embedding 이 잘못되었다는 점이다. Bilstm 은 음절단위로 분해되어 있는 반면 CNN 은 형태소 단위로 분석이 되는 것으로 판단이 되었으나 실제로는 한국어는 형태소 분석처리가 되지 않고 음절 처리로 되어 있었다. 따라서 해당하는 데이터가 없는 경우 CNN 은 올바르지 않는 답변을 하게 된다.
      i. 어절로 처리되어 발생하는 문제점은 학습되지 않은 어절이 들어가게 될 경우 새로운 어절로 생성된다는 점이다. 특히, Subject 정확도 파트에서도 기술하였지만 전체 데이터의 15%를 test 데이터로 사용하였으며 이를 분석하는 모델이다. 그러나 모든 데이터는 팀이 제작하였으며 팀 외부의 인물이 평가를 하기 위해서 집어넣은 문장들은 팀이 만드는 문장과 같다고 판단할 수가 없다.
      ii. 중간발표에서 데모로 테스트한 질문 중 "송강호가 태어난 곳은 어디인가?"라는 질문과 "송강호는 어디에서 태어났는가?" 라는 질문을 비교를 할 수 있다. 학습된 데이터는 "송강호가 태어난 곳은 어디인가?"라는 질문이지만 "송강호는 어디에서 태어났는가?"의 모든 어절은 학습된 데이터에는 전혀 없기 때문에 새로운 데이터로 입력, 이로 인하여 Predict Predicate 의 정확도는 떨어질 수 밖에 없다.
   D. Subject 와 Predicate 을 종합적으로 평가하여 SPARQL 데이터와의 매칭의 정확도를 평가하고자 한다. 이 때의 정확도는 26.3%로 매우 낮은 수치를 보이고 있다.
   E. Subject, Predicate 의 SPARQL 데이터와의 매칭의 정확도가 매우 떨어지므로 이 모델을 통해 정답률을 파악하는 것은 불필요하다고 판단이 되어 Question Answering 에 대한 정확도는 실시하지 않았다.

8. 개선사항

A. Simple Question Data 추가 생성

더욱 정확한 학습을 위하여 기존 데이터와 비교하여 데이터의 개수도 증가 시켰다. 기존 데이터는 Predicate 이 약 100 개이며 이에 대한 Question 총 개수는 약 800 개 였으나 기존 데이터에 새로운 데이터를 약 1000 개 가량 추가하여 총 데이터가 약 2000 개 가량 되도록 변경하였다.

B. Predict Subject model 의 변경

위에서 언급한 대로 BiLstm 과 BIO tagging 만 이용하여 모델을 구축하자 정확도가 너무 많이 낮은 모습을 보였다. 따라서 이 모델의 개선이 매우 시급하였으며 모델 향상을 위해 기존 모델에서 정확도를 높이기 위하여 여러가지의 기능을 추가해보았다.

1. BIO tagging 변경
   A. BIO 태깅 과정에서 'B'가 여러 개 나오는 것을 방지하기 위하여, 모델에 문장을 집어넣기 전 /S 라는 추가적인 tag 를 삽입해, 중복되는 B 가 등장하지 않도록 학습시키는 것을 목적으로 하였다.
      i. 실제로 이 작업을 거친 후 대부분의 문장에서 B 의 tagging 은 단 하나만 나오는 것을 확인할 수 있었다
   B. 입력 데이터를 완전히 거꾸로 입력하여 재학습을 시켰다.
      i. Input Question - 김대중이 태어난 곳은 어디인가?
      ii. Bi-LSTM Input = [?, O], [가, O], ... , [이, O], [중, I], [대, I], [김, B]
   C. BIO tagging 의 정확도는 96%로 5% 증가한 것을 볼 수 있다.
   D. 이 모델의 개체명의 인식인 Predict Subject 의 정확도는 70.2% 정도로 2 배가량 증가하였다.
   E.
2. Bigram Feature 이용
   A. 데이터베이스의 모든 Subject 와 Object 에 대해서 ['해리', '리포', '포터']의 형태로 분해된 데이터를 저장한다.
   B. BIO tagging 으로 개체명을 추출할 때, 추출된 데이터를 위의 방식과 동일하게 처리한다.
   C. A 에서 생성한 파일을 참조해, 가장 많은 pair 가 일치하는 Subject 엔티티를 실제 Predicted Subject 로 채택한다.
   D. 이 방식을 이용하면 결과값이 데이터베이스에 존재하는 값으로 한정되므로, String 의 100% 일치가 중요한 SPARQL 쿼리에 이용할 수 있게 된다.
      i. 이를 적용시켜 Subject 추출의 정확도를 확인한 결과 85.6%의 정확도를 보였다.

모델	Bilstm + BIO tagging (Base Model)	Base Model + BIO Augmentation	**Base Model + BIO Augmentation + Bigram features**
정확도	35.8	70.2	**85.6%**

<표 1. Subject 추출 모델의 개선에 따른 정확도 변화 추이>

C. Predict Predicate model 의 변경

Predicate Model 의 기존 모델에서 발생하는 문제점의 주요 원인은 CNN 모델이 음절이 아닌 어절 단위로 Word Embedding 이 되고 있기 때문이다. 따라서 Predicate 추출의 정확도를 상승시킬 수 있도록 이를 수정하고 하이퍼파라미터를 조금씩 튜닝하였다.

1. CNN 모델
   - A. 적합한 하이퍼파라미터를 찾기 위한 시도
     - i. Batch size, filtering data 등을 변경해 보았으나 효과는 미미
     - ii. Dropout ratio 를 0.5 에서 0.1 로 수정하자 정확도 상승 확인
   - B. 개선 결과 Predicate 추출의 평균 정확도는 약 87.5%로 상승

2. Train & Test 데이터 처리
   - A. 기본 데이터셋은 Subject 기준으로 정렬되어 있으나, 원활한 기계학습을 위해 이를 무작위로 셔플하였다.
   - B. Word Embedding 을 진행할 때, 어절 단위로 문장을 분해했기 때문에, 데이터베이스에 없는 단어가 Subject 로 선택될 경우 프로그램이 제대로 답을 이끌어 내지 못하는 것을 확인
     - i. 이에 대한 해결책으로 한국어 형태소 분석인 KoNLPy 를 이용하고자 했으나, 고유명사를 식별하지 못하는 문제가 발생
     - ii. Input Subject : 하정우는 …
     - iii. 형태소 분석 결과 : [하정] [우] [는] ….
   - C. 같은 의미이지만 미묘하게 다른 구성의 질문을 이해하지 못한다.
     - i. Train : 송강호의 출생지는 어디인가? 로 학습
     - ii. Test : 송강호의 출생지는? - 부산광역시
     - iii. Test : 송강호가 태어난 곳은 어디인가? – 결과 없음
   - D. 이를 해결하기 위해 Subject 추출과 동일한 솔루션을 사용, 어절 단위가 아닌 음절 단위로 분해된 문장을 입력받도록 CNN 모델을 수정하였다.
   - E. 개선 결과 Predicate 추출의 평균 정확도는 약 92%로 상승

모델	Base Model	Base Model+ Character Embedding	Base Model + Hyperparameter Tuning	**Base Model + Hyperparameter Tuning + Character Embedding**
정확도	76.8	92%	87.5	**92.9%**

<표 2. Predicate 추출 모델의 개선에 따른 정확도 변화 추이>

D. 개선 결과

모델의 정확도가 많이 개선된 것을 확인한 뒤, 위의 개선점들을 적용하여 최종적으로 KBQA 시스템을 구축하고 SPARQL 쿼리문을 생성해 데이터베이스에 질의를 넣을 수 있도록 프로그램을 구성하였다.

1. NL2SPARQL 의 정확도를 평가
   A. Predicted Subject 와 Predicted Predicate 모델의 정확도를 평가하며 NL2SPARQL 의 정확도를 평가하는 것과 같다.
   B. 평가 방식으로 Predicted Subject 와 Predicted Predicate 를 함께 평가하여 기존의 SPARQL 의 트리플 구조와 비교하여 완전히 일치하는지의 정확도를 평가한다.
   C.
2. Question Answering 의 정확도를 평가
   A. SPARQL 구조로 만들었기 때문에 Subject 와 Predicate 에 대응하는 Object 가 대한 답이 0 개(해당 데이터 없음) 혹은 1 개가 될 수 있으며, 복수 개의 정답이 존재하는 경우도 있다.
      i. 예를 들면 "송강호가 태어난 곳은 어디인가?"라는 질문에 대해서는 "대한민국"과 "부산 광역시" 2 가지의 정답이 존재한다.
      ii. 따라서 정확도를 측정하기 위하여 반환받은 모든 정답을 하나의 리스트에 저장을 해 두었다.
      iii. 기존 질문 문장의 Object 가 반환받은 정답 리스트에 존재하는지 확인하여 정확도를 평가한다.
   B. 정확도 측정 결과 전체 QA 시스템의 평균 정확도는 84.9%를 달성하였다.

9. UI 기능

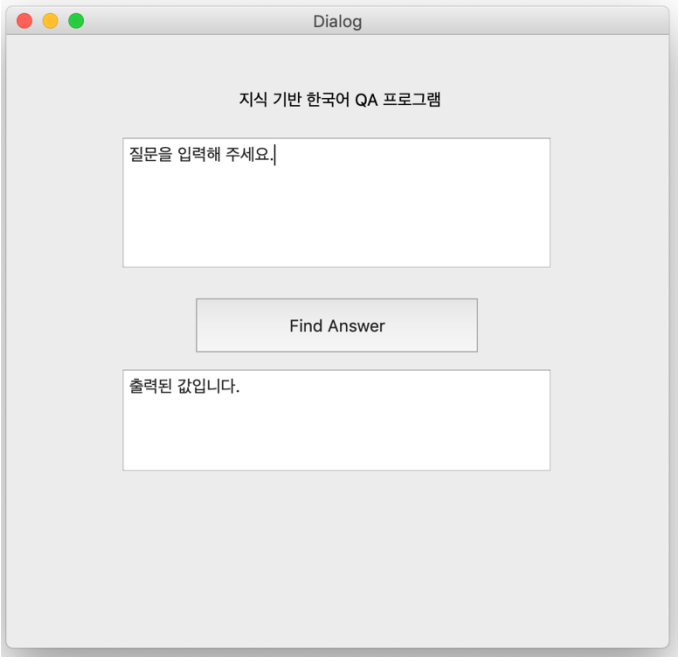
A. PyQT 를 이용한 UI 제작

사용자의 입장에서 프로젝트의 기능은 명확하다. 사용자가 질문을 입력하면 프로그램이 질문에 대한 대답을 반환하는 것이다. 따라서 UI 적 구성 요소로 질문을 입력하는 텍스트 박스, 프로그램에 질문을 입력값으로 넣는 버튼, 질문에 대한 대답, 즉 Object 값을 보여주는 텍스트 박스가 필요하다. GUI 적 요소를 위하여 QT 의 레이아웃에 Python 의 코드를 연결하여 GUI 프로그램을 만들 수 있게 해주는 프레임워크인 PyQT 를 사용하였다.

PyQT 를 이용할 때 프로그램의 레이아웃 편집을 용이하게 하는 편집기인 Qt Designer 을 활용하였다. Qt Designer 을 실행시키면 실제 프로그램 실행 시 화면 구성을 보여주는 Dialog Window, 버튼 등 각종 위젯 아이템들이 나열된 Widget Box, 위젯들의 속성을 설정할 수 있는 Property Editor 을 볼 수 있다. 사용자의 입력을 받는 텍스트 박스와 출력값을 디스플레이하는 텍스트 박스는 모두 QPlainTextEdit 속성을 사용하였고, Widget Box 에서 QPlainTextEdit 두개를 드래그하여 프로그램 구성 요소에 추가한 다음 Property Editor 에서 텍스트 박스들의 이름과 속성을 설정해주었다. 버튼의 경우 QPushButton 을 사용하였다. 입력값을 받는 텍스트 박스에는 '질문을 입력해 주세요.'라는 메시지를, 출력값을 반환하는 텍스트 박스에는 '출력된 값입니다.'라는 메시지를 각각 디폴트 값으로 설정하여 별도의 사용 설명 없이도 사용자가 직관적으로 프로그램을 사용할 수 있도록 유도하였다.

Qt Designer 에서 레이아웃 편집을 완료한 후 파이썬 프로그램 코드와 같은 디렉토리에 저장시키면 XML 형식의 .ui 파일이 생성된다. 파이썬 코드 내에서 PyQt5 를 import 하여 .ui 파일을 로드한 후 프로그램을 띄우는 클래스를 선언하고, 이 클래스 내에서 위젯들과 프로그램의 기능을 연결시킨다. 메인 함수에서 클래스의 인스턴스를 생성하여 .show() 와 .exec_()을 하게 되면 파이썬 프로그램에 UI 요소들이 포함되어 작동하는 것을 볼 수 있다.

사용자가 입력한 질문에 대한 대답이 DBConnector 을 통해 성공적으로 도출될 경우 출력값을 나타내는 텍스트 박스에 Object 반환값이 출력되며, 데이터의 부재 등의 이유로 반환값이 없을 경우 ':(' 가 출력된다. 또한 사용자는 프로그램을 매번 종료할 필요 없이 반복적으로 질문을 입력할 수 있다. 프로그램 실행 화면은 다음과 같다.



<그림 10. 프로그램 실행 화면>

실행 절차

Simple Question 을 입력하면 질문에 대한 대답을 반환하는 프로그램을 사용자가 직관적으로 사용할 수 있도록 PyQT 를 활용하여 제작

1. UI 를 생성하기 전에 Pickle 또는 tf.model 로 학습된 모델 호출
2. 모델 학습 후 UI 클래스의 인스턴스 생성
   A. '질문을 입력해 주세요.'라는 문장을 입력값을 받는 텍스트 박스의 디폴트 값으로 설정하여 사용자가 Simple question 을 이 곳에 입력하도록 유도함.
   B. 문장을 입력한 후 'Find Answer' 버튼을 누르게 되면 사용자의 입력값에 대한 대답이 답안 텍스트 박스에 반환됨

B. 답변으로 Subject 와 Predicate 으로 얻은 모든 데이터를 출력하며 만약 데이터의 부재 등의 이유로 반환값이 없을 경우 " :(" 을 출력함

10. 최종 데모

에이지 오브 울트론에 출연한 사람은 누구인가?

최종 데모를 위하여 Subject 에 해당하는 부분은 train 데이터에 존재하지 않는 데이터를 사용하였다. 또한 실제 사용자가 질문하는 어투를 반영하기 위하여 Subject 데이터 중 괄호나 특수 기호가 포함된 데이터의 일부는 제외하여 데모를 진행하였다. 예를 들어, "어벤져스 : 에이지 오브 울트론"의 경우 "에이지 오브 울트론"을 Subject 로 하여 질문을 생성해보도록 한다. 본 프로젝트의 취지에 맞도록 Simple question 생성시 Subject 가 문장의 시작점에 위치하도록 하였다. 다음 그림들은 프로그램의 실제 실행 화면에 해당한다.

실행 결과

Question 1 : 데이비트 베컴은 어디서 태어났는가?

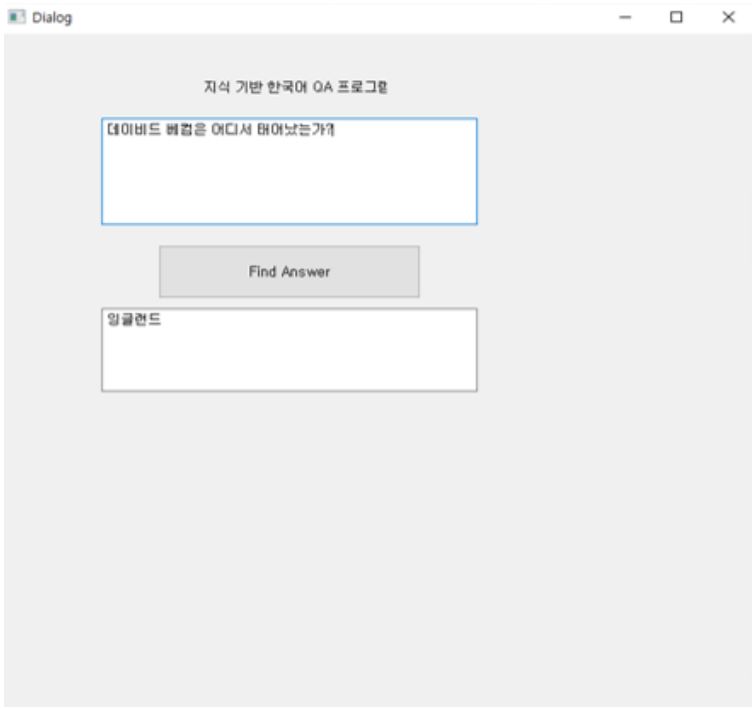
➔ 잉글랜드

Question 2 : 에이지 오브 울트론에 누가 출연했는가?

➔ Answer 2 : 스칼렛_요한슨

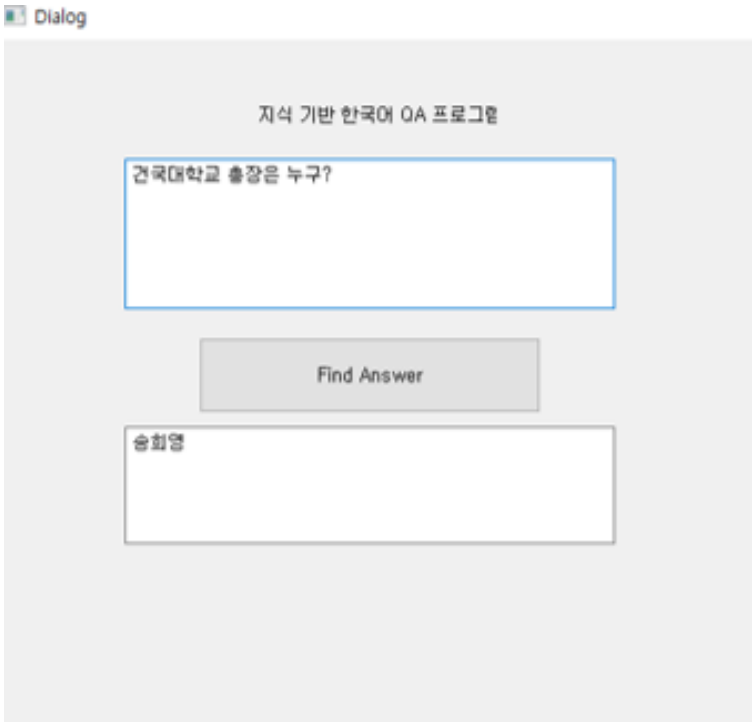
Question 3 : 건국대학교 총장은 누구?

➔ Answer 3 : 송희영



<그림 11. 데모 1: "데이비드 베컴은 어디서 태어났는가?">

<그림 12. 데모 2: “에이지 오브 울트론에 누가 출연했는가?”>



<그림 13. 데모 3: “건국대학교 총장은 누구?”>

11. 시스템 테스트 케이스

Case No.	테스트케이스 목표	입력 상황	예상 결과	실행 결과
TEST 1.1.1 (기능)	한국어로 된 시멘틱 트리플 데이터 데이터베이스를 구축해야 한다.	https://wiki.dbpedia.org/downloads-2016-10 로부터 다운받은 데이터를 기반으로 한국어 데이터베이스를 구축함	야나미_조지, birthPlace, 도쿄_시	성공 야나미_조지, birthPlace,도쿄_시
Test 1.2.1 (기능)	집계된 트리플 데이터 엔트리의 수가 30 만개 이상이어야 한다.	data_file = pd.read_csv('data0607/concat_ mappingbased_ objects_ko.csv') print(data_file.count())	300000 300000 300000	성공 523319 523319 523319
Test 1.2.2 (기능)	집계된 Subject 와 Object 종류가 각각 10 만개 이상이어야 한다.	print(len(subject_list)) print(len(object_list))	100000 100000	성공 113695 114485
Test 1.2.3 (기능)	집계된 Predicate 종류가 200 개 이상이어야 한다.	print(len(Predicate_list))	200	성공 244
TEST 2.1.1 (기능)	입력 쿼리에서 Subject 를 정확히 추출하는 Bi-LSTM 기반 모델이 구축되어야 한다.	야나미 조지가 태어난 곳은 어디인가?	"야나미_조지 "라는 데이터가 도출됨	성공 출력값 : "야나미_조지" 85.6%의 정확도

Test 2.1.2 (기능)	입력 쿼리에서 Predicate 을 정확히 추출하는 CNN 기반 모델이 구축되어야 한다.	야나미 조지가 태어난 곳은 어디인가?	"birthPlace" 이라는 데이터가 도출됨	성공 출력값 : "birthPlace" 92.9%의 정확도
TEST 2.2.1 (기능)	학습시킨 모델이 입력된 Simple Question 으로부터 정확히 Subject 와 Predicate 을 추출해야 한다.	사용자 입력: 야나미 조지가 태어난 곳은 어디인가?	Subject : 야나미 조지 Predicate : birthPlace	성공 Subject : 야나미 조지 Predicate : birthPlace
TEST 2.3.1 (기능)	Subject 와 Predicate 을 기반으로 Object 를 찾는 SPARQL 쿼리 스트링이 완성되어야 한다.	Subject : 야나미_조지 Predicated : birthPlace	Select birthPlace Where 야나미_조지	성공 {<http://ko.dbpedia.org/resource/야나미_조지> <http://dbpedia.org/ontology/birthPlace> ?b .} – SELECT ?b WHERE
TEST 2.3.2 (기능)	사전 구축한 데이터베이스를 참조하여 올바른 Object 값이 반환되어야 한다.	SELECT ?b WHERE { <http://ko.dbpedia.org/resource/야나미_조지> <http://dbpedia.org/ontology/birthPlace> ?b .}	도쿄시	성공 출력값 : "도쿄시" 84.9%의 정확도
Test 3.1 (비기능)	NL to SPARQL 과 SPARQL 질의응답 정답률을 각각 80% 이상 달성한다.	Simple Question 의 Test Dataset 을 모델에 입력	테스트 결과 각각 80% 이상의 정확도 달성	성공 정확도 지표 : NL to SPARQL Subject- 85.6% Predicate- 92.9% SPARQL 질의응답 84.9%

<표 3. 시스템 테스트 케이스>

12. 한계

프레임워크를 마지막까지 구현한 결과 최종적으로 84.9%라는 결과를 얻을 수 있었으나 아직 개선의 여지가 많은 모델이라고 판단되어, 추후 연구로 이를 해결하도록 한다. 현재 모델은 다음과 같은 한계를 가지고 있다.

1. 학습 데이터 부족
   A. Train data 로 사용할 한국어 Simple Question 데이터셋이 존재하지 않기 때문에, 인공지능 모델 학습에 사용할 데이터셋을 직접 제작해야 했다.
      i. 직접 제작한 데이터이기 때문에 문장을 생성하는 방식이 다채롭지 못하며, 이는 오버피팅을 일으킬 수 있다.
   B. 현재 모델에 적용한 솔루션 중 일부는 Simple Question 중에서도 Subject 가 문장의 맨 앞에 등장하는 경우에만 특출난 성능을 보인다.
      ii. 즉 문장의 중간에 Subject 가 등장할 경우 성능 하락이 있을 수 있다.
   C. 학습시키지 못한 데이터에 대한 질문은 대답할 수 없다.

<그림 14. 학습시키지 않은 데이터에 대한 질의 결과>

i. 위의 Subject 와 Predicate 은 질문을 "대한민국의 수도는 어디인가?"에 대해 추출되는 Subject 와 Predicate 이다. 우리의 상식으로는 답이 서울 이라는 것을 알고 있으나 SPARQL 데이터를 앞서 설명했듯이 일부만 사용했기 때문에 대한민국에 대한 Subject 에 capital 이라는 Predicate 에 대한 정답이 존재하지 않는다.

ii. 이를 해결하기 위하여 데이터셋의 양을 조금 더 늘리면 되겠지만 학습시간이 매우 커지게 되어 시간 내에 완료하지 못할 것으로 판단, 축소된 데이터로만 진행하였다.

iii.

2. 이음동의어 처리

A. Word Embedding 을 할 때, 존재하지 않는 단어 처리(DB 에 존재하나 Word2Vector 에 없는 단어들이 다수 존재)를 위하여 Word2Vector 를 하지 않고 One-Hot Embedding 방법을 사용했기 때문에 이음동의어 처리가 아예 불가능하다.

i. 예를 들어 한국이라는 Subject 를 뽑았으나 데이터셋에는 한국이 없으며 대한민국으로 통일되어 있다. 하지만 한국이라는 Subject 를 추출했기 때문에 한국의 데이터를 bigram 으로 유사도 분석을 하여 대한민국이 아닌 다른 subject 를 추출하게 된다.

B. 이를 처리하기 위하여 FastText 라는 word2Vector 를 사용할 수 있다.

i. FastText 를 이용하면 word2Vector 와 유사하게 사용할 수 있으며 더욱이 오타가 존재하면 FastText 가 자동으로 유사도 검사를 하여 원래 형태의 개체명을 제시한다.

ii. 사용해보고자 하였으나 FastText 의 데이터셋도 많을 뿐만이 아니라 다른 개선 사항을 이용하여 해결해보고자 하여 이 방식은 추후 연구로 미루도록 한다.

13. 향후 연구 계획

최종 정확도는 84.9%로 요구사항 기준치는 통과했지만 추가적인 연구를 통해 이를 더욱 개선할 수 있는 부분이 많이 보인다. 본 프로젝트에서는 Simple Question 으로 QA 를 진행하였으나 이를 강화해 복잡한 질문에 대해서 연구를 진행하거나, 정확도를 더욱 개선할 수 있을 것이라고 생각된다.

1. Simple Question 에서 복잡한 질문의 QA 처리
   A. Simple Question 은 트리플 구조 중 Subject 와 Predicate 으로 답변을 이끌어 내며 이 데이터들은 KB 데이터셋에 고스란히 저장이 되어 있어야 한다.
   B. 복잡한 질문은 Subject 와 Predicate 이 KB 데이터셋에 저장이 안되어 있지만 KB 데이터셋으로 유추가 가능을 할 수 있으며 그를 이용하여 새로운 Triple 구조를 생성할 수 있는 질문이다.
      i. "서울 시장의 나이는 몇 살인가?" 라는 질문에 대하여 나이를 통하여 age 를 유추할 수 있으나 서울 시장이 누구인지 확인을 해야한다. 따라서 "서울의 시장은 누구인가?"에 대한 답을 얻은 후 그 사람의 나이를 찾는 방식으로 진행되어야 한다.
      ii. 또 다른 특징으로 simple Question 은 학습할 때 맨 앞에 Subject 가 있게 학습이 되어 있다. 이는 학습데이터의 부족으로 판단이 되며 추후 학습을 할 때 이 점을 개선하여 해결할 수 있으면 좋다고 판단이 된다.
2. 정확도 및 컴퓨팅 파워 개선
   A. 정확도 개선을 위하여 사용한 방법이 기존 모델을 유지하면서 수치 또는 모델을 약간씩 변경하며 정확도를 높이는 방식을 선택했다.
   B. 다른 정확도 개선 방안으로 모델의 변경을 하는 방식도 존재한다.
      i. Subject 와 Predicate 을 분리된 개념이 아니라 통합된 개념으로 생각하여 한 번에 학습하는 방식이 존재할 수 있다.
      ii. 상기된 모델 중 하나인 Transformer 을 사용하여 한거번에 처리를 하게 되면 컴퓨팅 파워가 많이 축소될 것으로 판단된다.