**Korean educational reconjugator using NLP**

**경희대학교 컴퓨터공학과 캡스톤디자인1**

2015104203 이예준 2015104174 박기홍

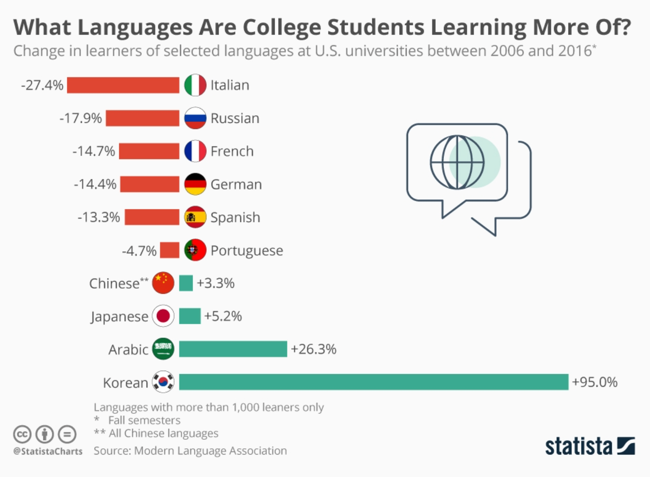
**요 약**

꾸준히 나타나는 한류 열풍으로 인해 외국인들의 한국어에 대한 관심이 높아지고, 한국어 배우기를 희망하는 외국인들이 많아졌다. 따라서 본 프로젝트에서는 이러한 외국인들을 대상으로 한국어 교육 서비스를 개발 중인 미리내 서비스들 중에서도 하나의 문장에 대해서 높임이나 시제, 평서문, 의문문 등으로 다양하게 문장을NLP 변환해주는 Korean reconjugator를 NLP를 이용하여 개발하고자 한다.

**1. 서론**

**1.1 연구배경**

최근 BTS가 빌보드 차트 1위를 기록하며 2012년도 싸이의 강남스타일 이래로 다시금 한류 열풍이 불고 있다. 이밖에도 봉준호 감독의 ‘기생충’이라는 영화가 오스카 4관왕을 휩쓸고, COVID-19로 인한 마스크 쓰기와 사회적 거리두기 역시 다른 나라들보다 상대적으로 뛰어난 시민의식을 보여주며 K-방역 등으로 한국의 위상이 높아지고 있다. [그림1]에서처럼 비단 최근의BTS나 ‘기생충’영화의 영향이 아니더라도 2006년도와 2016년도 사이에 미국 현대 언어 협회에 따르면 한국어 공부를 희망하는 학생들이 크게 늘어난 것을 확인할 수 있다.



[그림 1] Ratio of language learner between 2006-2016

하지만 한국어를 배우는 것은 그렇게 쉬운 일이 아니다. 발음면에서도 파열음으로 인해 어렵고 자음이 합쳐진 된소리나 특정 모음들 간의 발음 차이를 자국의 언어로 표현하지 못 하여 이를 인식하고 소리내기를 어려워한다. 또한 동음이의어나 어느 나라나 있다고는 해도 표현법에 있어서도 그 가짓수가 너무나도 다양하다.

따라서 미리내는 한국어에 관심이 있지만 진입장벽이 다소 높아 어려움을 겪는 외국인들을 대상으로 친숙하고 쉽게 공부할 수 있도록 돕는 서비스를 개발 중이다.

**1.2 연구목표**

미리내 서비스의 여러 기능들 중에서 하나의 문장에 대하여 높임이나 시제, 평서문, 의문문을 변환시켜주는 Korean reconjugator를 개발하고자 한다. Rule-based로 이 프로젝트를 해결하기엔 [그림 2]에서 보듯이 일반적으로 8개의 tense와 34개의 conjugation option들이 존재하고 이들 272개의 이외에도 현재 input으로 들어온 문장 자체의 서술부 어미의 형태와 현재의 conjugation type을 파악해야 하며 갖가지 탈락 규칙이 난무하는 한국어에 대하여 Rule-base를 적용하기란 쉽지 않은 방향이었다.

Table

Description automatically generated

[그림 2] Full options of reconjugator

따라서 NLP deep learning 통해 labeling된 한국어 말뭉치들을 가지고 여러 training-pairs를 만들고 이를 학습시켜서 스스로 conjugating 된 변환 결과를 예측해줄 수 있는 모델을 만들고자 한다.

**2. 관련 연구**

**2.1 기존 연구**

한국어 NLP에 관한 자료 중 우리가 서비스하려는 기능과 유사한 자료는 찾기가 힘들었다. 가장 유사한 프로젝트로 kakao i 를 찾아냈다. 기본 문체를 높임말과 예사말로 바꿔주는 기능이 존재했지만 주목적은 번역을 위해 만들어졌다. Input과 output의 형태를 고려했을 때 하나의 문장이 들어가고 변환된 문장이 나와야 했다. 이는 챗봇과 유사했고 이로부터 힌트를 찾았다. 하지만 자료 조사를 한 결과 챗봇을 개발할 때 보통 많은 양의 데이터를 요구하는 어려움으로 인해 대부분 open API를 이용하여 개발한 자료가 많았다. 그 중에 [그림 3]과 같이 직접 학습을 시키는 자료를 찾아냈는데Encoder와 Decoder구조를 이용하여 input sentence와 output sentence를 만들어 학습을 시키는 자료가 존재했다. 이 구조를 sequence to sequence 구조라 일컬으며 이는 [그림 4]에서처럼Deep learning을 이용한 번역 기계를 만드는 데에도 쓰였다.

Diagram

Description automatically generated

[그림 3] 한국어 Chatbot

Chart, box and whisker chart

Description automatically generated

[그림 4] Translator using sequence to sequence

**2.2 기존 연구의 문제점**

위에서 열거한 기존 연구들은 미리내에서 서비스하려는 기능을 구현하기 위해 구조적으로 유사한 연구 자료들이며 모두 맡은 기능들에 대하여 그에 적절한 데이터 셋을 갖추어 학습하였으며 그 성능도 뛰어난 것으로 파악된다. 해당 sequence to sequence model 자체에 문제점은 없다. 왜냐하면 여러 방면에서 그 성능이 인정이 되었기 때문에 이에 대한 Tutorial들도 다양하게 제시되어 있다. 문제는 서비스할 기능의 주목적에 맞추어 model을 활용하기 위해서Data preprocessing을 얼마나 잘 하느냐에 따라 달렸다.

**3. 프로젝트**

**3.1 기존 연구와 차이점 및 해결방안**

위에서 서술한 바와 같이 기존 연구에서 사용한 sequence to sequence model 자체에 문제는 없고 단순히 차이점만이 존재한다. 위 연구들에선 챗봇과 번역기능을 목표로 만들어졌기 때문에 그 데이터 셋의 내용과 형태가 그저 하나의 sequence라는 점에서만 같고 나머지는 전부 다르다고 볼 수 있다. 따라서 Reconjugator만을 위한training-pairs가 필요하다. 우리는 이 필요한 데이터 셋을 Mirinae pipeline을 통해서 추출한 뒤 새롭게 가공하여 사용했다.

또한 Pytorch에 내장되어 있는 default embedding method 를 버리고 OOV(Out of vocabulary)를 해결할 수 있는 gensim의 FastText를 사용하여 존재하는 모든 한국어 서술어에 대응할 수 있는 서비스를 만들고자 한다.

**3.2 프로젝트 내용**

**3.2.1 시스템 구성**

아래의 [그림 5]와 같이 Input으로 들어오는 모든 말뭉치들에 대하여 Mirinae pipelined을 거쳐 일련의 형태소 단위로 분해한다. 우리가 변환하고자 하는 부분은 서술부이다. 따라서 서술부 형태소를 중심으로 하여 각 서술부당 문장들을 묶어내고 해당 Tense나 Conjugating과 관련한 Labeling과정을 거친다. 그리고 문맥에 따라 Word embedding작업을 거치기 위한 Training data for FastText와 Reconjugating model의 학습을 위한 Training pairs를 따로 추출한다.

따라서 총 두 차례의 학습과정을 거치게 된다. 먼저 gensim의 FastText model을 통해 Word embedding 학습을 거쳐서 Word 즉, 형태소들에 대하여 Embedding vector들을 생성해주고 이를 Vocabulary에 dictionary 화 해준다. 그리고 각각의 형태소에 맞는 Embedding vector들을 sequence to sequence model의 초기 Embedding vector로 사용할 수 있도록 지정해준다. 그리고 sequence to sequence model을 통과하여 학습이 완료된다. 하지만 Output으로 나온 sequence는 당연히 앞서 형태소 단위로 분해했기 때문에 형태소의 sequence 이다. 이를 해결하기 위해서 Rule-based로 Morpheme merger를 자체적으로 개발하여 사용했다.

Diagram

Description automatically generated

[그림 5] Training system architecture

**3.2.2 Mirinae pipeline & Training Extractor**

일반적으로 미리내의 모든 서비스에 사용되는 데이터들은 Mirinae pipeline을 통과하게 된다. Pipeline의 여러 기능 중 Khaiii를 활용한 형태소 분석 과정을 거쳐 나온 일련의 문장 Sequence 들을 json파일에 담아 받을 수 있다. Reconjugating 개발을 위해 사용된 말뭉치들은 AI-hub와 KAIST corpus를 사용하였다. 특정 형태소들에 대한 Form-patterns를 분석해서 Labeling과정을 거쳐서 나온다. 이를 Training data로 가공하기 위해서 Extractor를 직접 개발하였다. 해당 Extractor를 통해서 두 가지 데이터셋을 뽑아내는데, 하나는 Word embedding vector를 만들기 위해 FastText를 학습시키는 데 필요한 데이터 셋과 sequence to sequence model에 학습시키기 위한 Training pairs를 추출해낸다. 최종적으로 Extractor를 통해 나온 데이터 셋은 아래 [그림 6]과 같다.

A picture containing text, receipt

Description automatically generated

[그림 6] Training data for FastText & sequence to sequence model

**3.2.3 Word embedding by FastText**

Word embedding을 하기 위한 method는 여러 개가 있다. 가장 일반적인 방법은 Pytorch에서 default로 제공하는 nn.embedding 방법이 있다. 해당 Embedding method는 word에 Indexing을 하고 sequence to sequence model의 학습 중에 Weight가 결정되는 방법이다. 그렇기 때문에 총 학습시간이800,000개 가량의 데이터로 진행했을 때 10시간 가량 넘어가는 문제가 있었고 Indexing과정을 거치지 않은 새로 등장하는 word 즉, 형태소에 관해선 제대로 예측하지 못 하는 문제점을 가지고 있었다.

따라서 첫 번째 문제인 시간 절약을 해결하기 위해서 Pre-trained embedding method를 도입해야 했다. 문서 수준에서의 Context가 고려되는 Embedding 기법 보다는 위의 [그림 6]에서와 같이 서술부나 짧은 문장에 있는 형태소 수준의 Embedding이 필요했기에 이를 충족시키는 Pre-trained embedding method인 Word2vec, Glove, FastText 등을 찾을 수 있었다.

그리고 마지막으로 Word embedding의 고질적인 문제인 OOV(Out of vocabulary)를 해결할 수 있는 method로 FastText가 선정되었다. 기존의 Word2vec이나 Glove는 학습 데이터 셋에 존재하지 않는 새로운 단어에 대해서 예측을 하지 못 하는 문제가 있다. 하지만 FastText 는 word 즉, 여기서 우리가 학습시킨 형태소를 Bag-of-character로 보기 때문에 하나의 형태소에 대하여 Character단위로 쪼개어 각 sub word에 대한 vector의 합을 계산하여 Embedding vector를 생산한다. 따라서 Morphological한 training 덕분에 training set에 존재하지 않는 word에 대해서도 그와 유사한 Embedding vector를 유추해낼 수 있는 것이다.

**3.2.4 Sequence to sequence model with Attention mechanism**

일반적으로 sequence to sequence model은 아래 [그림 7]과 같이 Encoder 와 Decoder 구조로 되어있다. 하지만 Encoder에서 나온 마지막 Context vector는 Encoder를 구성하는 마지막 RNN cell의 영향을 제일 크게 함축하고 있는 상태이기 때문에 만족스러운 결과를 얻지 못 할 수 있다.

Diagram

Description automatically generated

[그림 7] General Encoder & Decoder structure

그래서 고안된 방안이 Encoder의 Output인 Context vector가 거쳐온 각 RNN cell들에 대한 정보를 Decoder 예측에 있어서 고르게 반영될 수 있도록 도와주는 것이 아래의 [그림 8]에서 볼 수 있는 Attention mechanism이다.

Diagram

Description automatically generated

[그림 8] Attention mechanism

Encoder의 각 time-step마다 hidden state를 가져와 Decoder의 현재 time-step에 맞춰서 Attention weight를 구해주고 Context vector와 곱 연산을 진행하는 방식으로 이루어진다. 최종적으로 [그림 9], [그림 10]와 같이 Encoder와 Decoder 자체의 내부 cell인 RNN보다 성능이 더 좋은 LSTM을 간소화한 버전인 GRU를 사용하여 sequence to sequence model을 구성했다.

Diagram, timeline

Description automatically generated

[그림 9] Encoder structure

A picture containing diagram

Description automatically generated

[그림 10] Decoder structure attention applied

**3.2.5 Evaluate by Post fix & OOV Checker**

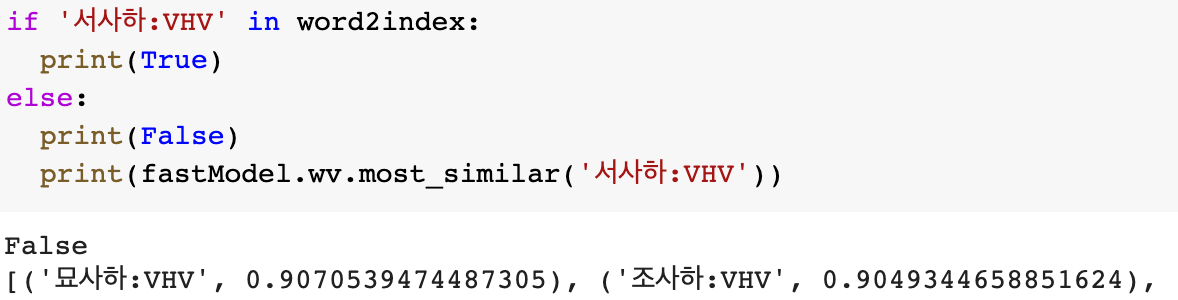
학습 모델의 안정적인 최종 설계를 계획하기 이전에 앞선 sequence to sequence model과 training data들을 통해 여러 차례 학습을 진행한 바 있다. 이때 보조용언이나 선어말 어미, 종결 어미 등은 대체적으로 높은 예측률을 보였다. 하지만 [그림 11]과 같이 앞의 용언부 즉, 학습 데이터로 들어오는 Verb phrase전반의 의미에 있어서 가장 큰 영향을 미치는 서술부에서도 오류가 나타났다. 다행히 FastText의 Bag of character로 생성된 Embedding vector로 인해 형태학적인 요소가 반영되어서 이후에 등장할 sequence의 어미 예측들에 크게 영향을 주진 않았다. 그렇다고 할지라도 용언부의 오류는 문장 의미에 큰 영향을 주기 때문에 이를 Fix할 수 있는 Algorithm을 개발하여 적용한다.

A picture containing text, receipt

Description automatically generated

[그림 11] Fix algorithm적용 이전의 예측 상태

또한 Vocab dictionary에 index와 함께 mapping되지 않은 새로운 word 즉, 형태소에 대하여 예측할 수 있도록 OOV 처리에 뛰어난 효율을 지닌 FastText의 장점을 최대한 살려서 OOV Checker method를 개발하여 적용한다. 아래 [그림 12]와 같이 형태학적으로 기존 Vocab dictionary에서 가장 유사도가 높은 형태소를 찾아내어 이로 대체시켜서 학습된Encoder와 Decoder를 통과시켜서 예측을 수행할 수 있도록 처리한다.



[그림 12] Measuring most similar word with OOV

**3.2.6 Morpheme merger system**

위 [그림11]의 ‘<’ 부분이 Output sequence를 나타내는데, 최종 결과값 역시 일련의 Morpheme sequence 로 출력된다는 것을 확인할 수 있다. 현재 개발하고자 하는 Reconjugator를 실제 서비스화 하기 위해서 꽤나 큰 중요도를 차지하고 있는 부분이기도 하다. 형태소 분석기는 Mirinae pipeline에서 사용한 카카오의 Khaiii말고도 여러 라이브러리들이 존재한다. 하지만 분해된 형태소를 다시 합치는 라이브러리는 자료조사에서 발견할 수 없었기 때문에 직접 개발하여 적용한다.

각각의 형태소에 Labeling된 Form pattern들과 여러 한국어 용언이나 접미사의 불규칙 등을 직접 분석해본 결과 ‘었’, ‘았’, ‘시’, ‘였’, ‘아’, ‘어’, ‘여’, ‘ㄴ’, ‘ㄹ’, ‘ㅂ’, ‘은’, ‘는’, ‘으’, 이 밖의 Special cases 등으로 카테고리화 할 수 있었다. [그림 13]과 같이 형태소와 형태소가 이어지는 부분을 Jamo-level로 Tokenize하여 각각의 접미사와 접두사가 만나는 과정에서 새롭게 생성되거나 탈락하는 자음과 모음을 Rule-base화 시켜서 개발하였다.

Diagram

Description automatically generated

[그림 13] ‘었’ 예외처리부의 ‘ㅂ’탈락 규칙의 예시

**3.2.7 Prototype for DEMO & Attaching to Mirinae service**

앞서 제시한 model을 통해 기존 Training-pairs의 90%를 학습으로 사용했고 10%를 테스트를 위해 사용한다. 이후 학습된 Encoder-Decoder model을 저장하고 미리내 서비스에 Merge하기 위한 Prototype을 개발하고자 한다. Prototype demo를 위한 서버는 아래의 [그림 14]과 같이Python의 FLASK를 이용해 구성하고 이곳에 학습된 model을 load한다. Output으로 나온 Morpheme sequence들은 직접 개발한 Merger를 통해 완성형 문장으로 재구성한다. Demo test를 위한 FRONT\_END는 React로 개발한다.

Diagram

Description automatically generated

[그림 14] DEMO Prototype architecture

**4. 프로젝트 결과**

**4.1 연구 결과**

위 프로젝트 설계 내용에서 서술한 바와 같이 Word embedding 과정은 전체 Training set에서의 모든 분해된 형태소에 대하여 FastText를 통해 학습시켰지만, sequence to sequence model의 학습 데이터는 1,050,226개의 pairs에서 90%만을 취하여 학습시켰고 10%를 테스트용으로 사용하였다. [그림 15]의 학습 곡선에서처럼 안정적으로 Loss가 줄어드는 것을 확인할 수 있었고 10%에서 랜덤하게 10개의 pairs를 꺼내어 테스트해본 결과는 [그림 16]과 같이 나온다.

Chart

Description automatically generated

[그림 15] Learning curve with loss

Text

Description automatically generated

[그림 16] Random 10 choices’ result of 10% test pairs

이후 10회의 학습 과정을 거치고 각각의 회차에서 10개씩 pairs를 가져와 총 100개의 sequence를 가지고 테스트하여 이를 분석해보았다.

|  |  |  |  |
| --- | --- | --- | --- |
| n | Correct | Same meaning | Incorrect |
| 1 | 8 | 2 | 0 |
| 2 | 8 | 1 | 1 |
| 3 | 5 | 3 | 2 |
| 4 | 8 | 2 | 0 |
| 5 | 8 | 2 | 0 |
| 6 | 8 | 2 | 0 |
| 7 | 8 | 0 | 2 |
| 8 | 8 | 1 | 1 |
| 9 | 8 | 1 | 1 |
| 10 | 5 | 2 | 3 |
| Total | 74 | 16 | 10 |

[표 1] Check stability of result

위의 [표 1]에서의 ‘Same meaning’은 완전히 똑같은 형태소를 예측한 것은 아니지만 의미 전달 상 큰 문제가 없는 경우이다. 예를 들어 ‘네:SEF’라는 종결어미로 예측 값이 기대되는 상황일 때 ‘네요:SEF’로 예측되는 경우 전체 문장의 의미 전달이나 reconjugating하는 데에 크게 문제가 되지 않고, 이러한 예측 값들을 분류하기 위해 만들어진 카테고리이다. 이렇듯 해당 카테고리의 경우에 대한 분류 함수화가 쉽지 않기 때문에 Training pairs 전체보다 적은 표본이지만 대체적으로 90%가량 의 정확성으로 Stability정도만을 파악할 수 있었다.

**4.2 Prototype web application**

추가적으로 이번 프로젝트를 미리내 서비스에 Attaching하고 또한 캡스톤 디자인1 Demo진행을 위해서 Web application형태의 Prototype을 개발하였다. 위의 프로젝트 설계 내용의 마지막부분과 같이 서버를 Flask를 통해 Running하였고 Client-side로는 ReactJS를 이용하여 개발했다. 아래 [그림17]과 같이 Conjugation 선택을 통해 Tense, Style, Form등 옵션을 지정할 수 있고 해당 옵션에 따라 [그림 18]처럼 Reconjugating 결과가 나타난다.

Graphical user interface, text, application

Description automatically generated

[그림 17] Conjugating options

Graphical user interface, application

Description automatically generated

[그림 18] Result of reconjugation

**5. 결론**

**5.1 기대효과**

이번에 미리내 서비스의 여러 기능들 중 하나인 Reconjugator의 개발을 담당하여 진행하게 되었다. 확실히 용언부가 포함된 하나의 문장을 외국인의 입장에서 배운다고 가정할 때, Tense나 문장 style, form에 따라서 다양하게 변형된 형태의 문장도 동시에 학습할 수 있어서 한국어 공부에 정말 큰 도움이 될 것이라고 생각한다.

무엇보다도 OOV(Out of vocabulary)까지 고려하여 설계된 모델이기 때문에, 어떠한 신조어가 와도 변환 문장을 만들어낼 수 있다는 점과 분해된 형태소들을 여러 탈락현상이나 예외를 기반으로 합쳐서 완성형 문장으로 Reconstruct할 수 있다는 점에서 완성도가 높다고 볼 수 있다.

**5.2 추후 연구 방향**

위 [표 1]에서 10회차에 걸친 학습 이후의 stability가 높고 적은 표본이지만 평균적으로 90% 가량의 정학도가 기대되지만 이를 더 개선하기 위해서 Encoder와 Decoder 학습에 쓰인Attention mechanism을 visualizing하여 각 형태소가 지정된 cell에서의 Attention score를 분석해보았다.

A picture containing chart

Description automatically generated

[그림 19] Attention visualization

OOV문제 해결과 분해된 형태소를 다시 합치는 method개발에 포커싱을 맞추어서 더 자세하게 찾아서 개발할 수는 없었지만 [그림 19]처럼 긴 Input sequence에 대해서 뒷부분에 대한 Attention이 matching이 잘 되지 못 하기 때문에 조금씩 오답을 도출했을 것이라고 예상된다. 뒷부분에 Focusing이 잘 되지 않는 이유는 Global attention mechanism을 도입해서이다. 가장 긴 sequence의 length를 기준으로 하여 Attention을 적용하기 때문에 그보다 작은 길이의 sequence에 한해서는 뒷부분에 Focusing이 덜 된다고 한다. 따라서 가변 길이의 sequence에 대해 Attention을 적용할 수 있는 local attention mechanism을 도입한다면 추후 성능 개선에 도움이 될 것이라고 생각한다.

**6. 참고 문헌**

[1] Ilya Sutskever, Oriol Vinyals, Quoc V. Le. (2014). Sequence to Sequence Learning with Neural Networks, arXiv:1409.3215v3.

[2] Zhuang Bairong, Wang Wenbo, Li Zhiyu, Zheng Chonghui, Takahiro Shinozaki. (2017). Comparative Analysis of Word Embedding Methods for DSTC6 End-to-End Conversation Modeling Track, in Proceedings of the

6th Dialog System Technology Challenges (DSTC6) Workshop.

[3] Minh-Thang Luong, Hieu Pham, Christopher D. Manning. (2015). Effective Approaches to Attention-based Neural Machine Translation, arXiv:1508.04025v5.

[4] Example of sequence to sequence:

<https://github.com/deepseasw/seq2seq_chatbot>

[5] Sequence to sequence reference:

<https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html>

[6] Statistic from MLA:

<https://www.statista.com/chart/19598/increase-in-students-enrolled-in-different-language-classes-at-us-universities/>

[7] Khaiii the morpheme analyzer:

<https://github.com/kakao/khaiii>

[8] FastText reference:

<https://radimrehurek.com/gensim/models/fasttext.html>

[9] AI-hub corpus:

<https://www.aihub.or.kr/aidata/87>

[10] KAIST corpus:

<http://semanticweb.kaist.ac.kr/home/index.php/KAIST_Corpus>