지식구조 2.0

- **Knowledge Graph Python 코드 설명**

❑ 사전 준비

- mecab을 쓰려면 몇가지 준비가 필요하다.
  https://wonhwa.tistory.com/49 이 사이트에 설명이 나와 있는데 C 폴더 안에 mecab이라는 폴더를 만들고 거기에서 mecab-ko-msvc-x64.zip이랑 mecab-ko-dic-msvc.zip을 넣고 압축을 풀어줘야한다. 이때 폴더 이름으로 파일을 만들고 그안에 압축해주면 안되고 mecab폴더 안에 바로 풀어줘야 한다.
  다 했는데도 만약에 경로를 인식을 못하면 https://joyhong.tistory.com/129 여기 아래에 수정하는 법이 나오는데 따라해보면 될 수도 있다.

  그리고 mecab이 인식하지 못하는 단어가 있을때 이를 사용자 사전에 추가해줘야 하는데 user_dictionary.ipynb를 사용하면 된다. 주석처리된것처럼 넣어주고 power shell에서 명령어를 실행해주면 반영된다. 구글에 mecab 사용자 사전 추가, 우선순위 설정 치면 잘 나온다.

  그 외에 requirement.txt에 install 해야할 리스트가 적혀 있다.

지식구조 2.0

- **Knowledge Graph Python 코드 설명**

❑ load_data.py

- reat_txt(name)

  txt 파일을 읽어오는 함수. data/txt 폴더 안에 abc.txt라는 파일이 있으면 name에 abc가 들어가고 이를 읽는다.
  f.readlines를 사용하게 되면 txt문서에서 new line을 기준으로 한줄씩 리스트 한칸을 차지하면서 읽어오기 때문에 paragraphs는 [‘첫째줄’, ‘둘째줄’, …] 의 형태를 가지게 됨.
  이때 첫째줄, 둘째줄은 단순히 문장1, 문장2가 되면 안되고 전체 문서가 있을때 그 문서에서 특정 주제로 쓰여지는 부분의 첫째부분, 둘째부분이 되어야함. 왜냐하면 이때 나누어 지는걸 기준으로 TF-IDF를 계산하기 때문에 최소한 문단 단위이거나 페이지 단위 아니면 그 이상이 되어야 한다.
  예를 들어 일기를 썼다고 하면 [“아침에 일어남. 그리고 어쩌고 저쩌고…“, “점심때 밥먹음. 축구도함. …. “, “수업들음. 저녁 뭐 먹음. 운동을 하러감. “] 이런 식으로 하나의 줄에 전체 글에서 하나의 부분이 되는 기준으로 잘라 져야함. 그래서 txt 데이터가 문장마다 newline 되어 있으면 TF-IDF가 잘 작동안 할 수도 있음

지식구조 2.0

- **Knowledge Graph Python 코드 설명**

❑ preprocess.py # 전처리와 관련된 함수들이 있음

- stopwords : 원하는 불용어를 넣어서 제거할 수 있음, clean_stopwords에서 사용됨

- mecab_tokenize(sent)

  한국어 형태소 분석기중 mecab이라는 형태소 분석기를 사용할 예정. 그 이유는 여러 tokenizer중에 가장 빠른 것으로 알려져 있음. sent는 말그대로 sentence임. “나는 걷는다”와 같은 문장이 오고 mecab.pos하면 mecab이 알아서 문장을 작은 요소로 나눈다 (ex. “나”, “는” “걷는다” 처럼 작은 부분으로 쪼갬). 그리고 그 중에 ‘NNG’와 ‘NNP’만 선택하는데 이게 ‘NNG’는 보통명사이고 ‘NNP’는 고유명사로 그냥 명사만 뽑아낸다고 보면 된다. 더 뽑고 싶으면 옵션을 더 넣으면 된다.

지식구조 2.0

- **Knowledge Graph Python 코드 설명**

❑ preprocess.py # 전처리와 관련된 함수들이 있음

- clean_txt(str)

  str속에 특수문자를 지우고 그 다음에 soynlp에서 제공하는 only_text를 사용하면 텍스트를 제외한 나머지 문자를 지운다. 자세한건 https://github.com/lovit/soynlp 참고

- clean_stopword(tokens)

  여기서 tokens는 mecab_tokenize를 거치고난 word들을 tokens라고 한다. 각 token에 대해 불용어 리스트(stopwords)에 들어있는지 확인하고 불용어 리스트에 들어있으면 빼고 아닌 것만 모아서 return함

- preprocessing(paragraph)

  이 함수는 paragraph을 input으로 받는다. 이때 paragraph이란 load_data.py의 read_txt에서 한줄한줄을 읽어서 리스트로 저장하고 있는데 이때의 하나의 줄을 의미한다. 이 한줄은 여러 문장으로 구성되어 있을 수도 있다. 어쨌든 각 줄에 대해서 clean_text와 mecab_tokenize, clean_stopword를 적용하고 최종 clean_tokens를 return한다.

지식구조 2.0

- **Knowledge Graph Python 코드 설명**

❑ preprocess.py # 전처리와 관련된 함수들이 있음

- preprocess_node(paragraphs)

  paragraphs는 read_txt의 결과물로 [‘첫째줄’, ‘둘째줄’, …]의 형태임. 각 줄마다 preprocessing을 적용해서 최종적으로는 [첫째줄로부터 전처리된 token들 list, 둘째줄에서 얻은 전처리된 token list, …] 의 형태이다. 즉, list(txt에서 newline으로 나눠진 paragraph)안에 list(각 줄에 해당하는 token list)가 있다.

- preprocess_edge(paragraphs)

  preprocess_node랑 비슷한데 얘는 문장 단위까지 나눠서 preprocessing한다고 보면 된다. 최종 결과는 [첫째줄의 문장1로부터 얻은 전처리된 token list, 첫째줄의 문장2로부터 얻은 전처리된 token list, …. , 둘째줄의 문장1로부터 얻은 전처리된 token list, ….] 이다.
  즉, list(txt의 newline 기준)안에 list(각 줄에서 문장(“. “)으로 나눔)안에 list(문장의 token list)이다.

지식구조 2.0

- **Knowledge Graph Python 코드 설명**

❑ preprocess.py # 전처리와 관련된 함수들이 있음

- make_dic_tfidf(list_of_wordlist)

  list_of_wordlist는 preprocess_node 함수의 결과로 list인데 각 요소가 token의 list로 이루어져있다.

  그걸 바탕으로 전체 word를 count를 세고(total_word_count) 또 각 word가 총 paragraph에 몇번 등장했는지도 세어서(word_frequency) tfidf를 계산하여 dictionary를 return한다.

- make_dic_count(tokens)

  이거는 위에서 tfidf 대신 그냥 Term-Frequency(그냥 몇번 등장했는지)만 구하는 함수이다.

- stcs_dic_count(ListOfSentence)

  문장 단위까지 Term-Frequency를 계산하는 함수.

지식구조 2.0

- **Knowledge Graph Python 코드 설명**

❑ graph.py # KnowledgeGraph class가 있고 관련 함수가 있음

- init

e_option : “scs”, “ss”, “pcs”, “ps” 중 하나 w_option : “TF-IDF”, “TextRank” 중 하나

scale : 1보다 큰 어떤수(1이 최소값이라 그럼) n_node : 노드 개수

r : cutting_edge 함수에서 사용됨, distance구할때 사용

- get_nodes

dictionary인데 TF-IDF/TextRank score가 높은순서대로 정렬된 { 노드1 : score1, 노드2 : score2, … , 노드n : score_n} 이 return 됨

이거는 위에서 tfidf 대신 그냥 Term-Frequency(그냥 몇번 등장했는지)만 구하는 함수이다.

지식구조 2.0

- **Knowledge Graph Python 코드 설명**

❑ graph.py # KnowledgeGraph class가 있고 관련 함수가 있음

- get_adj

  일단 cotable이라는 N x N 의 adjacency matrix를 return한다. 이 cotable은 symmetric해서 upper triangle만 채웠다. 이 cotable은 e_option에 따라서 계산하는 방식이 달라진다.

  “ss”와 “ps”는 각각 문장/문단을 기준으로 두 단어가 같은 문장/문단에 동시에 등장하면 cotable[i][j]값에 +1해준다. “scs”와 “pcs”는 문장/문단을 기준으로 i번째 token과 j번째 token의 vector를 구하고 둘의 cosine similarity를 비교하여 cotable[i][j]값이 채워진다. 그리고 util.rescale함수를 이용해 값을 1~scale 받은 값까지 정규화 해준다.

- make_graph

  앞에서 구한 node와 adj matrix를 바탕으로 networkx에서의 graph를 만들어 준다. 이때 노드이름이 “token이름 + “ “ + “score 점수” 로 되어 있는데 바꾸고 싶으면 바꿔도 된다.

지식구조 2.0

- **Knowledge Graph Python 코드 설명**

❑ graph.py # KnowledgeGraph class가 있고 관련 함수가 있음

- cutting_edge(graph, adj, r, option)

  앞에서 만든 graph의 edge는 대부분의 node가 edge가 있기 때문에 이를 보기 좋게 하려면 제거해줘야 한다. option은 cut하는 알고리즘을 받는데 “dijkstra”, “MST”, “PFNET”이 있다. 사실 dijkstra는 PFNET이랑 똑같아서 지워도 되는데 나중에 시간이 덜 걸리는걸 살리면 된다. MST는 딱히 안써서 지워도 되는데 일단 나누었다. 각 알고리즘은 dijkstra.py, kruskal.py, PFNET.py를 보면 된다.

- make_graph

  앞에서 구한 node와 adj matrix를 바탕으로 networkx에서의 graph를 만들어 준다. 이때 노드이름이 “token이름 + “ “ + “score 점수” 로 되어 있는데 바꾸고 싶으면 바꿔도 된다.

지식구조 2.0

- **Knowledge Graph Python 코드 설명**

❑ visualization.py # python dash를 이용하여 graph를 시각화함

- 사용방법

원하는 title을 visualization.py의 중간에 보면 title = “제목 어쩌구 … “ 에 넣어줘야 한다. 기본적으로 read_txt에서 data폴더를 보기 때문에 data/폴더1/제목1.txt에 있으면 “폴더1/제목1.txt”를 넣어주면 된다.
나머지 옵션들도 넣어주는데 가장 처음에 나오는 그래프가 거기에 설정해놓은 값으로 기본적으로 생성된다.

나머지는 python dash와 dash cytoscape를 보면서 공부하면 알 수 있다.
https://dash.plotly.com/ https://dash.plotly.com/cytoscape

지식구조 2.0

- **Knowledge Graph Python 코드 설명**

❑ util.py # 여러 계산 함수가 있다

- correlation(adj)

  그래프의 두 노드가 얼마나 다른 노드과 연관관계가 correlation이 있는지를 측정하는 함수이다. 이 값이 높으면 그래프의 노드들끼리의 상관관계가 높은거고 낮으면 낮은것이다. 자세한 계산법은 교수님이 주신 논문을 보면 된다.

- user_define_dict(dict)

  위 함수는 TF-IDF나 TextRank의 score 상위 N개를 노드로 뽑지 않고 그냥 내가 원하는 노드를 임의로 넣고 싶을때 사용한다. 원하는 노드를 주석처리된 것처럼 넣고 get_top_N에서 dict = user_define_dict(dict) 해주면 된다. TF-IDF나 TextRank 계산값을 무시하는 것이기 때문에 이때의 w_option은 의미 없다.